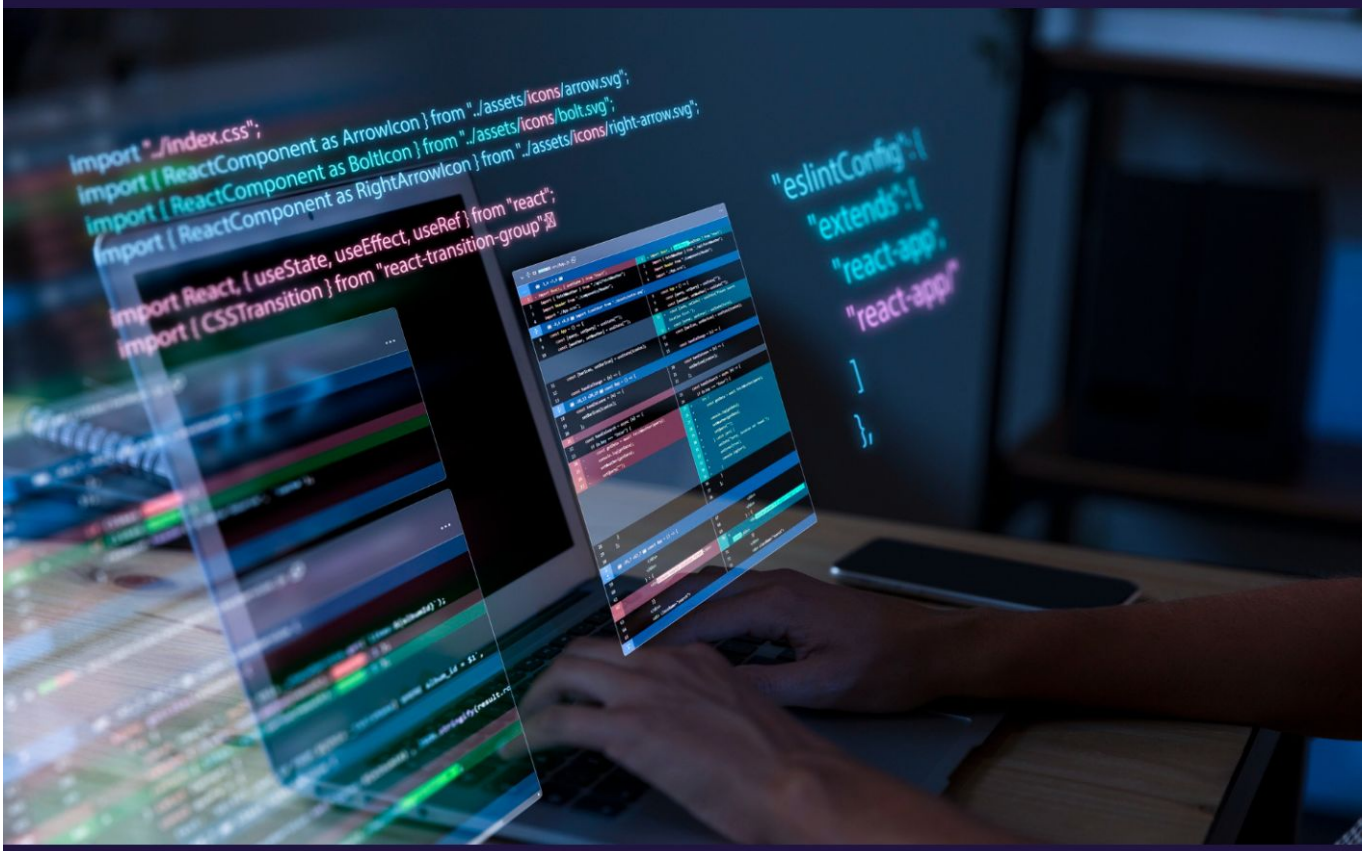
METODE KLASIFIKASI MODERN

DALAM MACHINE LEARNING MENGGUNAKAN PYTHON



Endar Nirmala, S.Kom., M.T.

METODE KLASIFIKASI MODERN DALAM MACHINE LEARNING MENGGUNAKAN PYTHON

Endar Nirmala, S.Kom., M.T.

MEDIA PUSTAKA INDO

2026

METODE KLASIFIKASI MODERN DALAM MACHINE LEARNING MENGGUNAKAN PYTHON

Penulis:
Endar Nirmala, S.Kom., M.T.

ISBN :
978-634-247-279-8

Penerbit :
PT MEDIA PUSTAKA INDO
Jl. Merdeka RT4/RW2
Binangun, Kab. Cilacap, Provinsi Jawa Tengah
No hp. 0838 6333 3823
Website: www.mediapustakaindo.com
E-mail: mediapustakaindo@gmail.com

Anggota IKAPI: 263/JTE/2023

Cetakan Pertama : 2026

KATA PENGANTAR

Puji syukur ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga buku yang berjudul "Metode Klasifikasi Modern dalam Machine Learning Menggunakan Python" ini dapat diselesaikan dengan baik. Buku ini disusun sebagai upaya untuk memberikan pemahaman yang komprehensif mengenai berbagai metode klasifikasi dalam machine learning serta implementasinya menggunakan bahasa pemrograman Python.

Perkembangan teknologi informasi dan data dalam beberapa dekade terakhir telah melahirkan berbagai inovasi dalam bidang kecerdasan buatan (*artificial intelligence*) dan *machine learning*. Salah satu teknik yang sangat penting dalam machine learning adalah metode klasifikasi, yaitu proses pengelompokan data ke dalam kategori tertentu berdasarkan pola atau karakteristik yang dimiliki oleh data tersebut. Metode ini banyak digunakan dalam berbagai bidang, seperti analisis data bisnis, pengenalan wajah, diagnosis medis, sistem rekomendasi, analisis sentimen, hingga keamanan siber.

Buku ini dirancang untuk memberikan pemahaman yang sistematis mengenai konsep dasar machine learning, khususnya metode klasifikasi modern. Pembahasan dalam buku ini mencakup berbagai algoritma klasifikasi yang banyak digunakan, seperti *Decision Tree*, *Naïve Bayes*, *Support Vector Machine*, *K-Nearest Neighbor*, hingga metode berbasis *ensemble learning*. Selain itu, buku ini juga menyajikan contoh implementasi praktis menggunakan Python dengan berbagai pustaka populer seperti *NumPy*, *Pandas*, *Scikit-learn*, dan *Matplotlib*, sehingga pembaca dapat memahami tidak hanya konsep teoretis, tetapi juga penerapannya dalam analisis data nyata.

Buku ini ditujukan bagi mahasiswa, peneliti, praktisi data, serta siapa saja yang tertarik mempelajari machine learning dan analisis data menggunakan Python. Materi disusun secara bertahap, mulai dari konsep dasar hingga penerapan metode klasifikasi secara praktis, sehingga diharapkan dapat membantu pembaca dalam memahami dan mengembangkan keterampilan di bidang analisis data dan kecerdasan buatan.

Akhir kata, semoga buku ini dapat memberikan manfaat dan kontribusi dalam pengembangan ilmu pengetahuan, khususnya dalam bidang machine learning dan analisis data, serta mendorong lahirnya inovasi berbasis teknologi data di masa depan.

DAFTAR ISI

BAB 1
Konsep Dasar Machine Learning dan Klasifikasi

1.1 Pengantar Machine Learning

Machine Learning (ML) adalah cabang dari kecerdasan buatan (Artificial Intelligence) yang berfokus pada pengembangan algoritma yang memungkinkan komputer belajar dari data dan membuat keputusan atau prediksi tanpa diprogram secara eksplisit untuk setiap aturan. Konsep ini berangkat dari gagasan bahwa data mengandung pola tersembunyi yang dapat dimodelkan secara matematis. Pada awal perkembangannya, sistem komputer bekerja berdasarkan aturan yang ditentukan secara manual (rule-based system). Namun, pendekatan ini memiliki keterbatasan ketika menghadapi data dalam jumlah besar dan pola yang kompleks. Machine Learning hadir sebagai solusi untuk memungkinkan sistem belajar secara otomatis dari pengalaman (data historis) [1]. Secara umum, Machine Learning bekerja dengan membangun model matematis yang memetakan input (fitur) ke output (target). Model ini dilatih menggunakan dataset sehingga dapat mengenali pola hubungan antara variabel. Setelah pelatihan selesai, model dapat digunakan untuk memprediksi data baru yang belum pernah dilihat sebelumnya. Konsep penting dalam Machine Learning adalah data. Data terdiri dari fitur (features) dan label (untuk supervised learning). Fitur adalah atribut atau karakteristik yang digunakan sebagai input model, sedangkan label adalah hasil yang ingin diprediksi. Kualitas data sangat memengaruhi performa model. Selain data, komponen penting lainnya adalah algoritma. Algoritma ML merupakan prosedur matematis yang digunakan untuk menemukan pola dari data. Contohnya termasuk Linear Regression, Logistic Regression, Support Vector Machine, dan Neural Network.

Machine Learning juga melibatkan konsep training dan testing. Data biasanya dibagi menjadi dua bagian: data latih

(training data) untuk melatih model dan data uji (testing data) untuk mengevaluasi performa model. Hal ini penting untuk menghindari overfitting. Overfitting terjadi ketika model terlalu menyesuaikan diri dengan data latih sehingga performanya buruk pada data baru. Sebaliknya, underfitting terjadi ketika model terlalu sederhana sehingga tidak mampu menangkap pola yang ada dalam data. Secara matematis, proses pembelajaran model dapat direpresentasikan sebagai optimisasi fungsi loss. Fungsi loss mengukur perbedaan antara prediksi model dan nilai aktual. Tujuan algoritma adalah meminimalkan fungsi loss tersebut. Sebagai contoh, pada regresi linear, fungsi loss yang digunakan adalah Mean Squared Error (MSE). Model mencoba menemukan parameter terbaik yang meminimalkan nilai MSE.

Machine Learning banyak digunakan dalam berbagai bidang. Di sektor kesehatan, ML digunakan untuk mendeteksi penyakit dari citra medis. Di bidang keuangan, ML digunakan untuk mendeteksi fraud. Di bidang pendidikan, ML dapat digunakan untuk memprediksi keberhasilan akademik mahasiswa. Salah satu kekuatan utama Machine Learning adalah kemampuannya menangani data dalam skala besar (big data). Dengan kemajuan komputasi dan ketersediaan GPU, model yang kompleks seperti Deep Learning dapat dilatih dengan dataset yang sangat besar. Machine Learning juga erat kaitannya dengan statistik dan probabilitas. Banyak algoritma ML didasarkan pada konsep statistik seperti distribusi probabilitas, estimasi parameter, dan inferensi statistik. Perkembangan Machine Learning tidak terlepas dari ketersediaan bahasa pemrograman yang mendukung komputasi ilmiah, seperti Python. Python menjadi bahasa yang sangat populer karena sintaksnya sederhana dan memiliki banyak library ML. Dalam praktiknya, Machine Learning bukan hanya tentang memilih algoritma terbaik, tetapi juga memahami data dan konteks masalah. Pemahaman domain sangat penting agar model yang dibangun relevan dan akurat. Machine Learning juga memiliki siklus kerja yang sistematis. Proses ini meliputi definisi masalah, pengumpulan data, preprocessing, pemodelan, evaluasi, dan deployment. Kemampuan interpretasi model juga menjadi

aspek penting. Beberapa model seperti Decision Tree lebih mudah diinterpretasikan dibandingkan model kompleks seperti Neural Network. Selain itu, etika dalam Machine Learning juga perlu diperhatikan. Model yang dilatih dengan data bias dapat menghasilkan keputusan yang tidak adil.

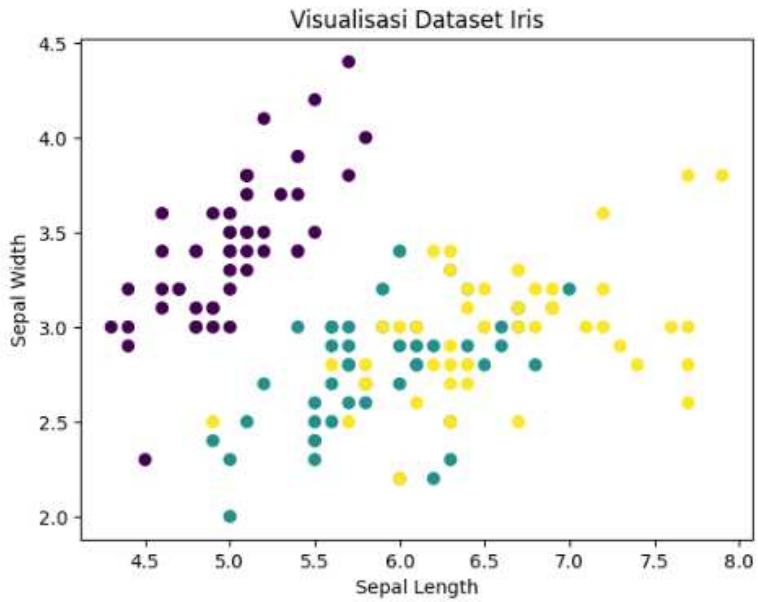
Dalam konteks klasifikasi, Machine Learning bertujuan mengelompokkan data ke dalam kategori tertentu. Contohnya adalah klasifikasi email spam dan non spam. Performa model biasanya diukur menggunakan metrik tertentu seperti accuracy, precision, recall, dan F1-score. Pemilihan metrik tergantung pada jenis masalah yang dihadapi. Proses pelatihan model melibatkan iterasi berulang untuk menemukan parameter optimal. Teknik seperti Gradient Descent digunakan untuk memperbarui parameter model secara bertahap. Machine Learning juga dapat dikombinasikan dengan teknik lain seperti feature engineering untuk meningkatkan performa model. Salah satu tantangan dalam Machine Learning adalah menangani data yang tidak seimbang (imbalanced dataset). Hal ini sering terjadi dalam kasus fraud detection. Keberhasilan implementasi Machine Learning tidak hanya ditentukan oleh algoritma, tetapi juga kualitas preprocessing dan validasi model. Seiring perkembangan teknologi, Machine Learning menjadi fondasi utama dalam pengembangan sistem cerdas modern seperti chatbot, sistem rekomendasi, dan kendaraan otonom.

Contoh Implementasi Dasar Machine Learning Menggunakan Python:

```
# Import library
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
```

```
# Load dataset Iris
data = load_iris()
X = data.data
y = data.target

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

# Membuat model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Prediksi
y_pred = model.predict(X_test)

# Evaluasi
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Visualisasi fitur pertama dan kedua
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.title("Visualisasi Dataset Iris")


plt.show()
```

```
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



1.2 Perbedaan Supervised, Unsupervised, dan Reinforcement Learning

Machine Learning secara umum dibagi menjadi tiga paradigma utama, yaitu Supervised Learning, Unsupervised Learning, dan Reinforcement Learning. Ketiga pendekatan ini memiliki karakteristik, tujuan, serta metode pembelajaran yang berbeda. Pemahaman terhadap perbedaan ketiganya sangat penting sebelum menentukan metode yang tepat untuk suatu permasalahan. Supervised Learning adalah pendekatan pembelajaran mesin yang menggunakan data berlabel. Artinya, setiap data input telah memiliki pasangan output atau target yang

diketahui. Model belajar dari hubungan antara input dan output tersebut untuk kemudian memprediksi output dari data baru. Pada supervised learning, dataset biasanya terdiri dari pasangan (X, y), di mana X adalah fitur dan y adalah label [2]. Tujuan algoritma adalah menemukan fungsi $f(X)$ yang mendekati nilai y. Contoh permasalahan supervised learning adalah klasifikasi email spam dan tidak spam, serta prediksi harga rumah. Supervised learning terbagi menjadi dua jenis utama, yaitu regresi dan klasifikasi. Regresi digunakan ketika target berupa nilai kontinu, sedangkan klasifikasi digunakan ketika target berupa kategori atau kelas diskrit. Contoh algoritma supervised learning antara lain Linear Regression, Logistic Regression, Decision Tree, Support Vector Machine, dan Random Forest. Semua algoritma tersebut memerlukan data berlabel untuk proses pelatihan.

Keunggulan supervised learning adalah performanya yang tinggi jika tersedia data berlabel dalam jumlah cukup. Namun, kelemahannya adalah proses pelabelan data seringkali mahal dan memakan waktu. Berbeda dengan supervised learning, Unsupervised Learning tidak menggunakan data berlabel. Model hanya diberikan data input tanpa target, dan tugasnya adalah menemukan struktur atau pola tersembunyi dalam data tersebut. Unsupervised learning sering digunakan untuk clustering dan dimensionality reduction. Clustering bertujuan mengelompokkan data berdasarkan kemiripan, sedangkan dimensionality reduction bertujuan mengurangi jumlah fitur tanpa kehilangan informasi penting. Contoh algoritma unsupervised learning adalah K-Means, Hierarchical Clustering, dan Principal Component Analysis (PCA). Pendekatan ini sangat berguna dalam eksplorasi data (exploratory data analysis). Keunggulan unsupervised learning adalah tidak memerlukan data berlabel, sehingga lebih fleksibel. Namun, interpretasi hasilnya seringkali lebih kompleks dibandingkan supervised learning. Reinforcement Learning (RL) berbeda secara fundamental dari dua pendekatan sebelumnya. RL berfokus pada agen yang berinteraksi dengan lingkungan untuk mencapai tujuan tertentu melalui sistem reward dan punishment.

Dalam reinforcement learning terdapat tiga komponen utama: agent, environment, dan reward. Agen mengambil tindakan dalam lingkungan, kemudian menerima reward sebagai umpan balik. Tujuan RL adalah memaksimalkan total reward kumulatif dalam jangka panjang. Pendekatan ini banyak digunakan dalam pengembangan game AI, robotika, dan sistem kontrol otonom. Konsep penting dalam RL adalah policy, yaitu strategi yang digunakan agen untuk menentukan tindakan berdasarkan keadaan (state). Policy dapat direpresentasikan sebagai fungsi yang memetakan state ke action. Selain policy, terdapat juga konsep value function yang mengukur seberapa baik suatu state dalam konteks reward jangka panjang. Algoritma seperti Q-Learning digunakan untuk memperkirakan nilai tersebut. Perbedaan utama antara supervised dan reinforcement learning terletak pada jenis umpan balik yang diterima. Pada supervised learning, umpan balik berupa label langsung, sedangkan pada RL umpan balik berupa reward yang seringkali tertunda. Sementara itu, perbedaan antara supervised dan unsupervised learning terletak pada keberadaan label. Supervised memiliki label, sedangkan unsupervised tidak. Dalam konteks aplikasi nyata, supervised learning sering digunakan dalam sistem prediksi dan klasifikasi. Unsupervised learning digunakan dalam segmentasi pelanggan dan analisis pola. Reinforcement learning digunakan dalam pengambilan keputusan dinamis. Pemilihan pendekatan tergantung pada jenis data dan tujuan masalah. Jika tersedia label, supervised learning biasanya menjadi pilihan utama. Jika tidak ada label, unsupervised learning lebih sesuai. Reinforcement learning digunakan ketika permasalahan melibatkan urutan keputusan dan interaksi dengan lingkungan. Ketiga pendekatan ini juga dapat dikombinasikan dalam praktik. Misalnya, unsupervised learning digunakan untuk preprocessing sebelum supervised learning diterapkan. Perkembangan Deep Learning juga memengaruhi ketiga paradigma ini. Deep Neural Network dapat digunakan dalam supervised, unsupervised, maupun reinforcement learning. Pemahaman konseptual mengenai perbedaan ketiga metode ini menjadi fondasi penting

dalam membangun sistem Machine Learning yang efektif. Dalam implementasi menggunakan Python, ketiga pendekatan ini memiliki library dan framework yang berbeda namun saling melengkapi.

Contoh Implementasi Supervised Learning (Klasifikasi)

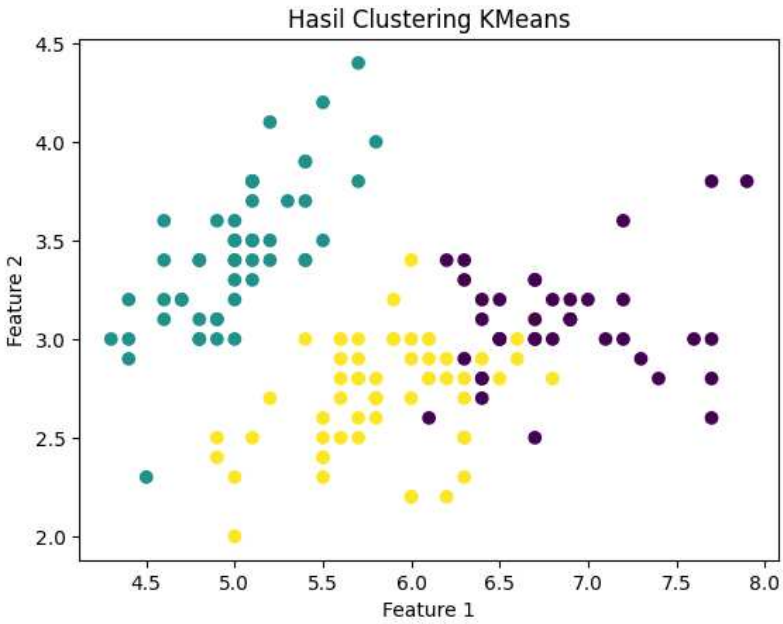
```
# Supervised Learning - Decision Tree Classifier

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X = data.data
y = data.target

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

# Model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Prediksi
y_pred = model.predict(X_test)
print("Akurasi Supervised Learning:", accuracy_score(y_test, y_pred))
```

```
Akurasi Supervised Learning: 1.0
```

Contoh Implementasi Unsupervised Learning (Clustering)

```
# Unsupervised Learning - KMeans Clustering

from sklearn.cluster import KMeans
```

```
import matplotlib.pyplot as plt

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

# Visualisasi cluster
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Hasil Clustering KMeans")
plt.show()
```



Contoh Implementasi Reinforcement Learning (Q-Learning Sederhana)

```
# Reinforcement Learning - Q-Learning sederhana

import numpy as np

# Inisialisasi Q-table
```

```
Q = np.zeros((5, 2))  # 5 state, 2 action
alpha = 0.1
gamma = 0.9

state = 0
action = 1
reward = 5

# Update Q-value
Q[state, action] = Q[state, action] + alpha * (reward + gamma *
np.max(Q[state]) - Q[state, action])

print("Q-table setelah update:\n", Q)
```

```
Q-table setelah update:
 [[0.  0.5]
 [0.  0. ]
 [0.  0. ]
 [0.  0. ]
 [0.  0. ]]
```

1.3 Konsep Dasar Klasifikasi

Klasifikasi merupakan salah satu tugas utama dalam supervised learning yang bertujuan untuk memetakan data input ke dalam kategori atau kelas tertentu. Berbeda dengan regresi yang menghasilkan nilai kontinu, klasifikasi menghasilkan output diskrit, seperti "Ya/Tidak", "Spam/Non-Spam", atau "Fraud/Non-Fraud". Dalam konteks machine learning modern, klasifikasi menjadi fondasi bagi banyak sistem cerdas. Secara matematis, klasifikasi berusaha membangun fungsi $f(x)$ yang memetakan fitur $x$ke label kelas y. Jika terdapat dua kelas, maka disebut klasifikasi biner [3]. Jika terdapat lebih dari dua kelas, disebut klasifikasi multikelas. Contoh sederhana klasifikasi biner adalah deteksi email spam. Model menerima fitur seperti jumlah kata tertentu, panjang email, atau keberadaan link, lalu menentukan apakah email tersebut spam atau bukan. Dalam klasifikasi multikelas, contoh umum adalah pengenalan jenis

bunga berdasarkan ukuran kelopak dan mahkota. Dataset Iris merupakan contoh klasik dalam studi klasifikasi multikelas. Konsep penting dalam klasifikasi adalah decision boundary. Decision boundary adalah garis atau bidang yang memisahkan kelas satu dengan kelas lainnya dalam ruang fitur. Algoritma klasifikasi bertujuan menemukan batas keputusan terbaik. Setiap algoritma memiliki cara berbeda dalam membentuk decision boundary. Logistic Regression membentuk batas linear, sedangkan Support Vector Machine dapat membentuk batas non-linear menggunakan kernel.

Dalam praktiknya, data klasifikasi sering kali memiliki banyak fitur. Oleh karena itu, ruang fitur dapat berdimensi tinggi. Visualisasi hanya mudah dilakukan pada dua atau tiga dimensi. Kinerja model klasifikasi diukur menggunakan berbagai metrik evaluasi. Accuracy adalah metrik paling umum, tetapi tidak selalu tepat untuk dataset tidak seimbang. Confusion matrix digunakan untuk menggambarkan performa model secara lebih detail. Matriks ini menunjukkan jumlah True Positive, True Negative, False Positive, dan False Negative. Precision dan recall menjadi penting ketika kesalahan prediksi memiliki konsekuensi berbeda. Misalnya, dalam deteksi penyakit, recall lebih penting agar kasus positif tidak terlewat. F1-score merupakan harmonisasi antara precision dan recall. Metrik ini berguna ketika ingin keseimbangan antara keduanya. Dalam klasifikasi probabilistik seperti Logistic Regression, model tidak hanya memberikan prediksi kelas, tetapi juga probabilitas keanggotaan suatu kelas. Threshold probabilitas dapat disesuaikan untuk mengontrol sensitivitas model. Secara default, threshold biasanya 0.5 untuk klasifikasi biner. Permasalahan lain dalam klasifikasi adalah imbalanced dataset, yaitu ketika jumlah data antar kelas tidak seimbang. Hal ini sering terjadi pada kasus fraud detection. Teknik seperti oversampling, undersampling, dan SMOTE sering digunakan untuk mengatasi ketidakseimbangan data. Klasifikasi juga dapat bersifat multilabel, di mana satu data dapat memiliki lebih dari satu label sekaligus. Dalam dunia nyata, klasifikasi digunakan dalam sistem rekomendasi, analisis sentimen, pengenalan wajah, dan diagnosis

medis. Proses klasifikasi selalu diawali dengan preprocessing data, seperti normalisasi, encoding, dan feature selection. Model klasifikasi dapat dibagi menjadi model linear dan non-linear. Model linear lebih sederhana dan mudah diinterpretasikan. Model non-linear seperti Random Forest dan Neural Network mampu menangkap pola kompleks tetapi lebih sulit diinterpretasikan. Pemilihan algoritma tergantung pada ukuran dataset, kompleksitas pola, dan kebutuhan interpretasi. Cross-validation sering digunakan untuk memastikan model tidak overfitting. Regularisasi seperti L1 dan L2 digunakan untuk mengurangi kompleksitas model dan mencegah overfitting. Evaluasi performa model juga dapat dilakukan menggunakan ROC curve dan AUC. Secara keseluruhan, klasifikasi merupakan teknik yang sangat penting dalam machine learning karena banyak permasalahan nyata berbentuk kategorisasi.

Contoh Implementasi Klasifikasi Menggunakan Python:

1. Klasifikasi Menggunakan Logistic Regression

```
# Import library
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# Load dataset
data = load_iris()
X = data.data
y = data.target

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

# Model Logistic Regression
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
```

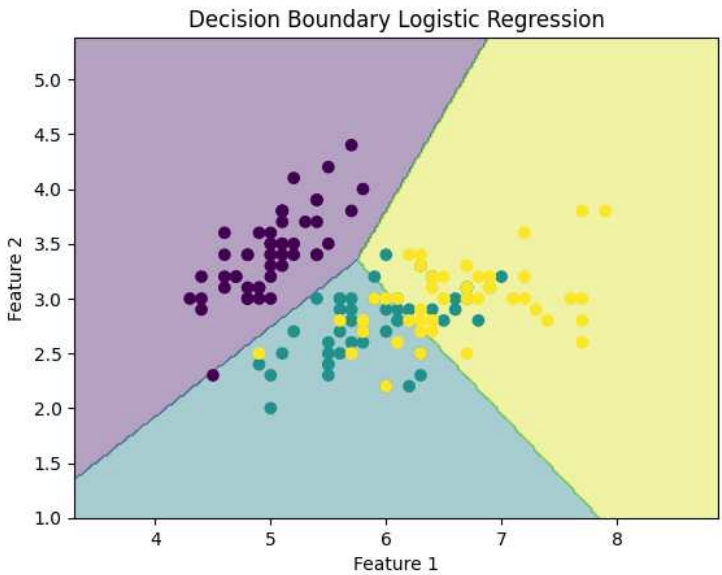
```
# Prediksi
y_pred = model.predict(X_test)

# Evaluasi
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 1.0

Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

2. Visualisasi Decision Boundary (2 Fitur)

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

# Gunakan hanya 2 fitur untuk visualisasi
X2 = X[:, :2]
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y, test_size=0.3, random_state=42)

model2 = LogisticRegression(max_iter=200)
model2.fit(X_train2, y_train2)
```

```
# Buat grid
x_min, x_max = X2[:, 0].min() - 1, X2[:, 0].max() + 1
y_min, y_max = X2[:, 1].min() - 1, X2[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

Z = model2.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X2[:, 0], X2[:, 1], c=y)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Decision Boundary Logistic Regression")
plt.show()
```



3. Klasifikasi Menggunakan Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("Accuracy Random Forest:", accuracy_score(y_test, y_pred_rf))
```

```
Accuracy Random Forest: 1.0
```

1.4 Tahapan Umum Proyek Machine Learning

Dalam praktiknya, membangun sistem Machine Learning tidak hanya sekadar memilih algoritma dan melatih model. Terdapat tahapan sistematis yang harus dilakukan agar model yang dihasilkan akurat, stabil, dan dapat digunakan dalam lingkungan nyata. Tahapan ini sering disebut sebagai Machine Learning Lifecycle. Secara umum, tahapan proyek Machine Learning terdiri dari [4]:

a. Problem Definition

Tahap pertama dalam proyek Machine Learning adalah mendefinisikan masalah secara jelas. Tanpa definisi masalah yang tepat, model yang dibangun tidak akan relevan dengan kebutuhan bisnis atau penelitian. Pada tahap ini, perlu ditentukan apakah permasalahan termasuk klasifikasi, regresi, clustering, atau reinforcement learning. Misalnya, jika ingin memprediksi apakah transaksi termasuk fraud atau tidak, maka permasalahan tersebut termasuk klasifikasi biner. Selain menentukan jenis masalah, penting juga menentukan metrik evaluasi yang akan digunakan. Dalam kasus deteksi fraud, recall mungkin lebih penting dibandingkan accuracy karena kita tidak ingin melewatkan transaksi fraud. Tahap ini juga mencakup pemahaman domain (domain knowledge). Pemahaman konteks bisnis membantu menentukan fitur yang relevan dan risiko yang mungkin muncul.

b. Data Collection

Data merupakan komponen utama dalam Machine Learning. Tanpa data yang memadai, model tidak dapat belajar

dengan baik. Data dapat diperoleh dari berbagai sumber seperti database internal, API, sensor IoT, web scraping, atau dataset publik seperti Kaggle dan UCI Repository. Kualitas data sangat penting. Data harus representatif terhadap kondisi nyata. Jika data bias, maka model juga akan bias.

Contoh membaca dataset menggunakan Python:

```
import pandas as pd

# Membaca dataset CSV
data = pd.read_csv("data.csv")

print(data.head())
print(data.info())
```

c. Data Preprocessing

Data mentah seringkali tidak siap langsung digunakan untuk pelatihan model. Terdapat missing values, outlier, atau data kategorikal yang perlu dikonversi. Tahap preprocessing mencakup pembersihan data (data cleaning), normalisasi, encoding, dan handling missing values. Preprocessing bertujuan meningkatkan kualitas data sehingga model dapat belajar dengan optimal.

Contoh preprocessing sederhana:

```
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import numpy as np

# Contoh data dengan missing value
X = np.array([[1, 2], [np.nan, 3], [7, 6]])

# Mengisi missing value
imputer = SimpleImputer(strategy="mean")
X_clean = imputer.fit_transform(X)

# Normalisasi
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_clean)
```

```
print(X_scaled)
```

d. Exploratory Data Analysis (EDA)

EDA dilakukan untuk memahami karakteristik data sebelum pemodelan. Pada tahap ini dilakukan analisis statistik deskriptif, distribusi data, korelasi antar fitur, serta visualisasi. EDA membantu mengidentifikasi pola, anomali, atau ketidakseimbangan kelas.

Contoh EDA sederhana:

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

# Statistik deskriptif
print(data.describe())

# Visualisasi distribusi
data.hist(figsize=(8,6))
plt.show()

# Korelasi
sns.heatmap(data.corr(), annot=True)
plt.show()
```

e. Feature Engineering

Feature engineering adalah proses membuat atau memilih fitur yang paling relevan untuk model. Tahap ini dapat meningkatkan performa model secara signifikan. Contoh feature engineering termasuk membuat fitur baru dari kombinasi fitur lama atau melakukan feature selection.

Contoh feature selection:

```
from sklearn.feature_selection import SelectKBest, chi2

X = data.drop("target", axis=1)
y = data["target"]

selector = SelectKBest(score_func=chi2, k=3)
X_selected = selector.fit_transform(X, y)
```

```
print("Shape setelah feature selection:", X_selected.shape)
```

f. Model Selection dan Training

Setelah data siap, tahap berikutnya adalah memilih algoritma yang sesuai. Pemilihan model tergantung pada jenis masalah, ukuran dataset, dan kebutuhan interpretasi. Model kemudian dilatih menggunakan data training.

Contoh training model:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

X = data.drop("target", axis=1)
y = data["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier()
model.fit(X_train, y_train)
```

g. Model Evaluation

Setelah model dilatih, perlu dilakukan evaluasi menggunakan data testing. Evaluasi bertujuan mengukur kemampuan generalisasi model. Metrik yang digunakan tergantung pada jenis masalah.

Contoh evaluasi:

```
from sklearn.metrics import accuracy_score, classification_report

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

h. Hyperparameter Tuning

Setiap algoritma memiliki hyperparameter yang dapat dioptimalkan untuk meningkatkan performa. GridSearchCV

dan RandomizedSearchCV sering digunakan untuk mencari kombinasi parameter terbaik.

Contoh tuning:

```
from sklearn.model_selection import GridSearchCV

param_grid = {
   "n_estimators": [50, 100],
   "max_depth": [None, 5, 10]
}

grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best Parameter:", grid.best_params_)
```

i. Deployment

Setelah model siap, tahap selanjutnya adalah deployment agar dapat digunakan dalam sistem nyata. Deployment dapat dilakukan melalui API, web service, atau integrasi ke aplikasi. Model dapat disimpan menggunakan joblib.

```
import joblib

joblib.dump(model, "model.pkl")

# Load kembali
loaded_model = joblib.load("model.pkl")
```

j. Monitoring dan Maintenance

Setelah deployment, model harus dimonitor performanya. Data di dunia nyata dapat berubah (data drift), sehingga model perlu diperbarui. Monitoring memastikan model tetap akurat dan relevan.

1.5 Tools dan Library Python untuk Machine Learning (NumPy, Pandas, Matplotlib, Scikit-learn)

Dalam pengembangan Machine Learning menggunakan Python, keberhasilan implementasi sangat dipengaruhi oleh

pemilihan tools dan library yang tepat. Python menjadi bahasa pemrograman paling populer dalam bidang data science karena memiliki ekosistem library yang lengkap, stabil, dan mudah digunakan. Library Python dirancang untuk mendukung komputasi numerik, manipulasi data, visualisasi, hingga pemodelan machine learning secara efisien. Empat library utama yang paling sering digunakan adalah NumPy, Pandas, Matplotlib, dan Scikit-learn. NumPy (Numerical Python) merupakan fondasi komputasi numerik dalam Python [5]. Library ini menyediakan struktur data array multidimensi dan berbagai fungsi matematika tingkat lanjut. Hampir semua library machine learning berbasis Python bergantung pada NumPy. Keunggulan utama NumPy adalah performanya yang tinggi dalam operasi matriks dan vektor. Dalam machine learning, hampir seluruh data direpresentasikan dalam bentuk matriks. Array NumPy lebih efisien dibandingkan list Python karena disimpan dalam blok memori yang berdekatan. Hal ini memungkinkan operasi matematika dilakukan secara cepat menggunakan vectorization.

Berikut contoh penggunaan NumPy:

```
import numpy as np

# Membuat array
data = np.array([[1, 2, 3], [4, 5, 6]])

# Operasi statistik
print("Mean:", np.mean(data))
print("Standar Deviasi:", np.std(data))

# Operasi matriks
print("Transpose:\n", data.T)
```

Selain operasi dasar, NumPy mendukung operasi aljabar linear seperti dot product yang sangat penting dalam algoritma machine learning.

```
# Dot product
A = np.array([[1, 2], [3, 4]])
```

```
B = np.array([[5, 6], [7, 8]])

print("Dot Product:\n", np.dot(A, B))
```

Setelah NumPy, library penting berikutnya adalah Pandas. Pandas digunakan untuk manipulasi dan analisis data dalam bentuk tabel (DataFrame). Dalam proyek machine learning, sebagian besar dataset berbentuk tabel. Pandas menyediakan struktur data DataFrame yang memudahkan filtering, grouping, agregasi, dan pembersihan data.

Contoh penggunaan Pandas:

```
import pandas as pd

# Membuat DataFrame
df = pd.DataFrame({
   'Nama': ['Andi', 'Budi', 'Citra'],
   'Nilai': [80, 75, 90],
   'Lulus': [1, 0, 1]
})

print(df.head())
print(df.describe())
```

Pandas juga mendukung pembacaan berbagai format file seperti CSV, Excel, dan JSON.

```
# Membaca file CSV
# df = pd.read_csv("data.csv")

# Filtering data
lulus = df[df['Lulus'] == 1]
print(lulus)
```

Dalam tahap preprocessing, Pandas sering digunakan untuk menangani missing values.

```
# Menambahkan missing value
df.loc[1, 'Nilai'] = None
```

```
# Mengisi missing value
df['Nilai'].fillna(df['Nilai'].mean(), inplace=True)
print(df)
```

Library berikutnya adalah Matplotlib. Visualisasi data merupakan bagian penting dalam exploratory data analysis (EDA). Matplotlib memungkinkan pembuatan grafik seperti histogram, scatter plot, dan line chart. Visualisasi membantu memahami distribusi data, korelasi antar fitur, serta pola tertentu.

Contoh penggunaan Matplotlib:

```
import matplotlib.pyplot as plt

nilai = [80, 75, 90, 85, 70]

plt.hist(nilai)
plt.title("Distribusi Nilai")
plt.xlabel("Nilai")
plt.ylabel("Frekuensi")
plt.show()
```

Scatter plot sering digunakan untuk melihat hubungan antara dua variabel.

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.scatter(x, y)
plt.title("Scatter Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

Visualisasi decision boundary dalam klasifikasi juga menggunakan Matplotlib. Library inti dalam machine learning adalah Scikit-learn. Scikit-learn menyediakan berbagai algoritma klasifikasi, regresi, clustering, dan evaluasi model. Scikit-learn

dirancang dengan API yang konsisten: model dibuat, dilatih dengan .fit(), dan digunakan untuk prediksi dengan .predict().

Contoh klasifikasi menggunakan Scikit-learn:

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

# Load dataset
data = load_iris()
X = data.data
y = data.target

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

# Model
model = RandomForestClassifier()
model.fit(X_train, y_train)

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

Scikit-learn juga menyediakan preprocessing tools seperti StandardScaler dan LabelEncoder.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Selain itu, tersedia fitur cross-validation untuk evaluasi model yang lebih robust.

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print("Cross-validation scores:", scores)
```

Pipeline dalam Scikit-learn memungkinkan penggabungan preprocessing dan model dalam satu alur.

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC

pipeline = Pipeline([
   ('scaler', StandardScaler()),
   ('svm', SVC())
])

pipeline.fit(X_train, y_train)
print("Pipeline Accuracy:", pipeline.score(X_test, y_test))
```

BAB 2
Persiapan dan Preprocessing Data

2.1 Pengumpulan dan Pemahaman Dataset

Pengumpulan dan pemahaman dataset merupakan tahap awal yang sangat krusial dalam proyek Machine Learning. Kualitas data yang digunakan pada tahap ini akan sangat menentukan keberhasilan model yang dibangun pada tahap-tahap selanjutnya. Prinsip umum dalam Machine Learning menyatakan "*garbage in, garbage out*", yang berarti model sebaik apa pun tidak akan menghasilkan prediksi yang baik jika data yang digunakan berkualitas rendah [6]. Pengumpulan dataset adalah proses memperoleh data yang relevan dengan permasalahan yang ingin diselesaikan. Data tersebut harus merepresentasikan kondisi nyata dari domain masalah. Misalnya, dalam kasus klasifikasi kelulusan mahasiswa, data yang dikumpulkan seharusnya mencerminkan kondisi akademik mahasiswa secara riil, bukan data sintetis yang tidak realistis. Sumber dataset dalam proyek Machine Learning sangat beragam. Data dapat diperoleh dari database internal organisasi, sistem informasi akademik, transaksi keuangan, sensor IoT, log sistem, hasil survei, maupun dataset publik. Setiap sumber data memiliki karakteristik, kelebihan, dan keterbatasan masing - masing.

Dataset publik sering digunakan dalam penelitian dan pembelajaran karena mudah diakses dan telah terstandarisasi. Namun, dataset publik tidak selalu sesuai dengan kebutuhan spesifik suatu masalah. Oleh karena itu, peneliti perlu memahami konteks dataset tersebut sebelum menggunakannya. Dalam pengumpulan data, penting untuk memastikan bahwa data yang dikumpulkan bersifat relevan. Data relevan adalah data yang memiliki hubungan langsung atau tidak langsung dengan target yang ingin diprediksi. Mengumpulkan terlalu banyak data yang tidak relevan justru dapat meningkatkan kompleksitas tanpa meningkatkan performa model. Selain relevansi, jumlah data juga menjadi faktor penting. Secara umum, semakin banyak data yang tersedia, semakin baik kemampuan model untuk belajar pola.

Namun, jumlah data yang besar harus diimbangi dengan kualitas data yang baik agar tidak menimbulkan noise berlebihan. Setelah data dikumpulkan, tahap selanjutnya adalah pemahaman dataset. Pemahaman dataset bertujuan untuk mengenali struktur, isi, dan karakteristik data sebelum dilakukan preprocessing dan pemodelan. Tahap ini sering disebut sebagai *data understanding*. Pemahaman dataset dimulai dengan mengidentifikasi tipe data yang dimiliki. Data dapat berupa numerik, kategorikal, teks, citra, atau time series. Setiap tipe data memerlukan teknik pengolahan yang berbeda dalam Machine Learning. Selain tipe data, perlu juga dipahami struktur dataset, seperti jumlah baris (instance), jumlah kolom (fitur), serta peran masing-masing kolom. Pada supervised learning, salah satu kolom berperan sebagai label atau target, sedangkan kolom lainnya berperan sebagai fitur. Analisis statistik deskriptif merupakan bagian penting dari pemahaman dataset. Statistik seperti mean, median, standar deviasi, nilai minimum, dan maksimum memberikan gambaran awal mengenai distribusi data. Informasi ini membantu mendeteksi ketidakwajaran pada data.

Pemahaman dataset juga mencakup identifikasi nilai hilang (*missing values*). Nilai hilang dapat muncul karena kesalahan input, keterbatasan sistem pencatatan, atau data yang tidak tersedia. Jika tidak ditangani dengan baik, missing values dapat menurunkan performa model. Selain missing values, outlier atau nilai ekstrem juga perlu diperhatikan. Outlier dapat muncul akibat kesalahan pencatatan atau memang mencerminkan kondisi yang jarang terjadi. Penentuan apakah outlier harus dihapus atau dipertahankan bergantung pada konteks permasalahan. Distribusi data juga perlu dianalisis pada tahap ini. Distribusi yang tidak seimbang, terutama pada variabel target, dapat menyebabkan model bias terhadap kelas mayoritas. Hal ini sering terjadi pada kasus klasifikasi fraud atau penyakit langka. Pemahaman dataset juga melibatkan analisis hubungan antar fitur. Korelasi antar fitur dapat memberikan informasi penting tentang redundansi data. Fitur yang sangat berkorelasi mungkin tidak perlu digunakan semuanya karena dapat menyebabkan multikolinearitas. Dalam konteks Machine Learning, pemahaman dataset tidak hanya

bersifat teknis tetapi juga konseptual. Peneliti perlu memahami makna setiap fitur dalam konteks domain masalah. Tanpa pemahaman domain, interpretasi hasil model bisa keliru. Dataset yang baik seharusnya mencerminkan kondisi dunia nyata secara adil dan tidak bias. Bias dalam data dapat muncul akibat metode pengumpulan yang tidak representatif, misalnya hanya mengumpulkan data dari kelompok tertentu.

Tahap pemahaman dataset juga membantu dalam menentukan strategi preprocessing yang tepat. Dengan memahami karakteristik data, peneliti dapat menentukan metode normalisasi, encoding, atau feature engineering yang sesuai. Dalam proyek Machine Learning yang berskala besar, dokumentasi dataset menjadi hal yang penting. Dokumentasi mencakup sumber data, metode pengumpulan, periode waktu, serta keterbatasan dataset. Dokumentasi ini membantu memastikan reprodusibilitas penelitian. Pemahaman dataset juga memudahkan komunikasi antara tim teknis dan pemangku kepentingan non-teknis. Dengan pemahaman yang baik, hasil analisis dapat dijelaskan secara lebih transparan dan dapat dipertanggungjawabkan. Kesalahan pada tahap pengumpulan dan pemahaman dataset sering kali baru terlihat pada tahap akhir proyek. Oleh karena itu, investasi waktu dan perhatian pada tahap awal ini sangat penting untuk menghindari kesalahan yang mahal di kemudian hari.

2.2 Data Cleaning dan Handling Missing Values

Data cleaning merupakan proses pembersihan data mentah agar siap digunakan dalam analisis dan pemodelan Machine Learning. Tahap ini sangat penting karena data yang diperoleh dari dunia nyata hampir selalu mengandung berbagai permasalahan, seperti nilai hilang, duplikasi, inkonsistensi, dan noise. Tanpa proses data cleaning yang baik, performa model Machine Learning dapat menurun secara signifikan. Tujuan utama data cleaning adalah meningkatkan kualitas data sehingga data tersebut lebih akurat, konsisten, lengkap, dan relevan. Data yang bersih memungkinkan algoritma Machine Learning belajar pola yang sebenarnya, bukan pola yang terbentuk akibat kesalahan atau ketidakteraturan data. Salah satu permasalahan paling umum

dalam data cleaning adalah missing values atau nilai yang hilang. Missing values terjadi ketika suatu fitur tidak memiliki nilai pada satu atau lebih instance data. Kondisi ini dapat muncul karena kesalahan input, kegagalan sensor, perbedaan format data, atau data yang memang tidak tersedia.

Missing values tidak boleh diabaikan begitu saja. Banyak algoritma Machine Learning tidak dapat bekerja dengan baik jika terdapat nilai kosong, bahkan beberapa algoritma tidak dapat dijalankan sama sekali. Oleh karena itu, penanganan missing values menjadi bagian krusial dari data cleaning. Sebelum menangani missing values, langkah pertama yang perlu dilakukan adalah mengidentifikasi keberadaan dan pola missing values. Peneliti perlu mengetahui fitur mana yang memiliki nilai hilang, berapa jumlahnya, dan apakah pola kehilangan data bersifat acak atau sistematis. Missing values secara umum dapat dikategorikan menjadi tiga jenis, yaitu *Missing Completely at Random (MCAR)*, *Missing at Random (MAR)*, dan *Missing Not at Random (MNAR)* [7]. Pemahaman jenis missing values membantu menentukan strategi penanganan yang paling tepat. Pada kasus MCAR, data hilang secara acak dan tidak bergantung pada variabel lain. Pada kondisi ini, penghapusan data sering kali tidak menimbulkan bias yang signifikan. Namun, pada kasus MAR dan MNAR, penghapusan data dapat menyebabkan bias serius pada model. Salah satu teknik paling sederhana dalam menangani missing values adalah menghapus data. Penghapusan dapat dilakukan pada level baris (row deletion) atau kolom (column deletion). Teknik ini hanya disarankan jika jumlah missing values sangat kecil dan tidak memengaruhi distribusi data secara signifikan. Penghapusan baris data dapat menyebabkan berkurangnya jumlah dataset secara drastis jika missing values cukup banyak. Hal ini berisiko menurunkan kemampuan generalisasi model, terutama jika dataset awal berukuran kecil.

Alternatif yang lebih umum digunakan adalah imputasi, yaitu menggantikan missing values dengan nilai tertentu. Teknik imputasi yang paling sederhana adalah mengganti missing values dengan nilai statistik seperti mean, median, atau modus. Imputasi

menggunakan mean sering digunakan untuk data numerik yang berdistribusi normal. Namun, metode ini sensitif terhadap outlier. Sebaliknya, median lebih robust terhadap outlier sehingga sering menjadi pilihan yang lebih aman. Untuk data kategorikal, imputasi biasanya dilakukan dengan mengganti missing values menggunakan nilai modus atau kategori yang paling sering muncul. Metode ini sederhana tetapi tetap harus digunakan dengan hati-hati agar tidak menimbulkan bias. Selain imputasi sederhana, terdapat teknik imputasi yang lebih canggih, seperti imputasi berbasis model. Pada metode ini, nilai yang hilang diprediksi menggunakan algoritma Machine Learning berdasarkan fitur lain yang tersedia. Contoh metode imputasi berbasis model antara lain K-Nearest Neighbor Imputation dan regresi. Teknik ini cenderung lebih akurat, tetapi memerlukan komputasi yang lebih kompleks. Data cleaning juga mencakup penanganan data duplikat. Data duplikat dapat muncul akibat kesalahan penggabungan data atau proses pencatatan ganda. Keberadaan data duplikat dapat menyebabkan model terlalu menekankan pola tertentu yang sebenarnya tidak signifikan. Selain itu, data cleaning melibatkan pemeriksaan inkonsistensi data, seperti perbedaan format penulisan, satuan yang tidak seragam, atau kesalahan penamaan kategori. Inkonsistensi ini harus diseragamkan agar data dapat diproses secara konsisten. Outlier atau nilai ekstrem juga merupakan bagian dari permasalahan data cleaning. Outlier dapat berasal dari kesalahan input atau memang mencerminkan kondisi yang jarang terjadi. Keputusan untuk menghapus atau mempertahankan outlier harus mempertimbangkan konteks domain.

Dalam konteks Machine Learning, data cleaning bukan sekadar tahap teknis, tetapi juga tahap analitis. Peneliti perlu memahami implikasi setiap keputusan pembersihan data terhadap interpretasi dan performa model. Penanganan missing values yang tidak tepat dapat menyebabkan model menjadi bias, overfitting, atau underfitting. Oleh karena itu, setiap teknik yang digunakan harus didokumentasikan dengan jelas. Data cleaning yang baik juga meningkatkan reproducibility penelitian. Peneliti lain dapat

memahami bagaimana data diproses dan mereplikasi hasil yang diperoleh. Dalam proyek Machine Learning skala besar, proses data cleaning sering kali memakan waktu paling lama dibandingkan tahap lainnya. Namun, investasi waktu ini sangat sepadan dengan peningkatan kualitas hasil akhir. Tools seperti Python dengan library Pandas dan Scikit-learn sangat membantu dalam proses data cleaning dan handling missing values. Library tersebut menyediakan fungsi yang efisien dan mudah digunakan.

Contoh Data Cleaning dan Handling Missing Values:

1. Membuat Dataset Contoh dengan Masalah Data : Dataset berikut mensimulasikan data akademik mahasiswa yang mengandung missing values**,** inkonsistensi, dan duplikasi.

```
import pandas as pd
import numpy as np

# Membuat dataset contoh
data = pd.DataFrame({
   'nama': ['Andi', 'Budi', 'Citra', 'Dina', 'Budi'],
   'nilai': [80, np.nan, 90, 75, np.nan],
   'kehadiran': [90, 85, np.nan, 70, 85],
   'jurusan': ['Informatika', 'informatika', 'Sistem Informasi',
None, 'informatika'],
   'lulus': [1, 1, 1, 0, 1]
})
```

	nama	nilai	kehadiran	jurusan	lulus
0	Andi	80.0	90.0	Informatika	1
1	Budi	NaN	85.0	informatika	1
2	Citra	90.0	NaN	Sistem Informasi	1
3	Dina	75.0	70.0	None	0
4	Budi	NaN	85.0	informatika	1

```
print(data)
```

2. Identifikasi Missing Values : Langkah pertama adalah mengidentifikasi jumlah dan lokasi missing values.

```
# Mengecek missing values
print(data.isnull().sum())
```

```
nama          0
nilai         2
kehadiran     1
jurusan       1
lulus         0
dtype: int64
```

3. Menghapus Data Duplikat
   # Menghapus data duplikat

```
    nama  nilai  kehadiran           jurusan  lulus
0   Andi   80.0       90.0       Informatika      1
1   Budi    NaN       85.0       informatika      1
2  Citra   90.0        NaN  Sistem Informasi      1
3   Dina   75.0       70.0              None      0
```

   data = data.drop_duplicates()
   print(data)

4. Menangani Missing Values – Data Numerik
   a. Imputasi Mean (Nilai Rata-rata) : Digunakan jika data relatif berdistribusi normal.

```
0    80.000000
1    81.666667
2    90.000000
3    75.000000
Name: nilai, dtype: float64
```

      data['nilai'].fillna(data['nilai'].mean(), inplace=True)
      print(data['nilai'])

   b. Imputasi Median (Lebih Robust terhadap Outlier)

```
0    90.0
1    85.0
2    85.0
3    70.0
Name: kehadiran, dtype: float64
```

```
data['kehadiran'].fillna(data['kehadiran'].median(),
inplace=True)
print(data['kehadiran'])
```

5. Menangani Missing Values – Data Kategorikal
   a. Imputasi Modus

```
# Mengisi jurusan kosong dengan nilai yang paling sering
muncul
data['jurusan'].fillna(data['jurusan'].mode()[0],
inplace=True)
print(data['jurusan'])
```

6. Menangani Inkonsistensi Data : Menyamakan format teks agar konsisten.

```
# Menyeragamkan huruf
```

```
0         informatika
1         informatika
2    sistem informasi
3         informatika
Name: jurusan, dtype: object
```

```
data['jurusan'] = data['jurusan'].str.lower()
print(data['jurusan'])
```

7. Validasi Hasil Data Cleaning

```
print(data)
```

```
    nama      nilai  kehadiran           jurusan  lulus
0   Andi  80.000000       90.0       informatika      1
1   Budi  81.666667       85.0       informatika      1
2  Citra  90.000000       85.0  sistem informasi      1
3   Dina  75.000000       70.0       informatika      0

Missing Values setelah cleaning:
nama         0
nilai        0
kehadiran    0
jurusan      0
lulus        0
dtype: int64
```

```
print("\nMissing Values setelah cleaning:")
print(data.isnull().sum())
```

2.3 Encoding Data Kategorikal

Encoding data kategorikal merupakan salah satu tahap penting dalam preprocessing data pada proyek Machine Learning. Data kategorikal adalah data yang nilainya berupa kategori atau label teks, seperti jenis kelamin, jurusan, status pernikahan, atau kategori produk. Sebagian besar algoritma Machine Learning tidak dapat memproses data dalam bentuk teks secara langsung, sehingga data kategorikal harus diubah menjadi representasi numerik. Tujuan utama encoding data kategorikal adalah mengubah kategori menjadi angka tanpa menghilangkan makna informasi yang terkandung di dalamnya. Proses ini harus dilakukan dengan hati-hati, karena metode encoding yang tidak tepat dapat menyebabkan model salah menafsirkan hubungan antar kategori. Data kategorikal secara umum dibagi menjadi dua jenis, yaitu nominal dan ordinal [8]. Data nominal adalah data kategori yang tidak memiliki urutan tertentu, seperti warna atau jenis jurusan. Sebaliknya, data ordinal memiliki urutan atau tingkat tertentu, seperti tingkat pendidikan atau tingkat kepuasan. Perbedaan antara data nominal dan ordinal sangat penting dalam menentukan teknik encoding yang tepat. Penggunaan metode encoding yang tidak sesuai dapat memperkenalkan hubungan palsu antar kategori yang sebenarnya tidak ada.

Salah satu teknik encoding paling sederhana adalah Label Encoding. Pada metode ini, setiap kategori diberikan nilai numerik unik. Misalnya, kategori “Informatika”, “Sistem Informasi”, dan “Teknik Komputer” dapat diberi label 0, 1, dan 2. Label Encoding cocok digunakan untuk data ordinal karena mempertahankan urutan kategori. Namun, untuk data nominal, Label Encoding dapat menimbulkan masalah karena model dapat menganggap adanya hubungan numerik antar kategori yang sebenarnya tidak bermakna. Contoh penggunaan Label Encoding sering dijumpai pada fitur seperti tingkat pendidikan (SMA < D3 < S1 < S2). Dalam kasus ini, representasi numerik mencerminkan urutan yang logis.

Teknik encoding yang paling umum untuk data nominal adalah One Hot Encoding. Pada metode ini, setiap kategori diubah menjadi kolom biner terpisah yang bernilai 0 atau 1. One-Hot Encoding menghindari asumsi hubungan ordinal antar kategori. Namun, metode ini dapat menyebabkan peningkatan jumlah fitur secara signifikan, terutama jika jumlah kategori sangat banyak. Fenomena ini dikenal sebagai *curse of dimensionality*. Dalam dataset dengan kategori yang sangat banyak, One-Hot Encoding dapat meningkatkan kompleksitas model dan memperlambat proses training. Oleh karena itu, teknik ini harus digunakan dengan bijak.

Alternatif lain adalah Ordinal Encoding, yang secara khusus digunakan untuk data ordinal. Teknik ini memetakan kategori ke angka berdasarkan urutan yang telah ditentukan sebelumnya. Ordinal Encoding memungkinkan model memahami hubungan tingkat antar kategori, tetapi membutuhkan pengetahuan domain untuk menentukan urutan yang benar. Selain metode dasar, terdapat teknik encoding lanjutan seperti Binary Encoding dan Hash Encoding. Teknik ini sering digunakan untuk dataset dengan jumlah kategori yang sangat besar. Binary Encoding menggabungkan Label Encoding dan representasi biner untuk mengurangi jumlah kolom dibandingkan One-Hot Encoding [9]. Metode ini lebih efisien dalam penggunaan memori. Hash Encoding menggunakan fungsi hash untuk memetakan kategori ke sejumlah fitur tetap. Metode ini sangat efisien, tetapi berpotensi menimbulkan collision, yaitu dua kategori berbeda memiliki representasi yang sama. Encoding data kategorikal juga harus memperhatikan potensi data leakage. Encoding harus dilakukan setelah pemisahan data training dan testing agar informasi dari data testing tidak bocor ke data training. Dalam konteks supervised learning, encoding dapat memengaruhi performa model secara signifikan. Model berbasis jarak seperti KNN dan SVM sangat sensitif terhadap representasi numerik data. Beberapa algoritma seperti Decision Tree dan Random Forest relatif lebih toleran terhadap encoding tertentu, tetapi tetap membutuhkan data dalam bentuk numerik. Encoding data kategorikal juga berperan penting dalam pipeline Machine Learning. Dengan menggunakan pipeline,

encoding dapat digabungkan dengan preprocessing lain secara konsisten.

Dalam praktik nyata, pemilihan metode encoding sering kali bersifat eksperimental. Peneliti perlu mencoba beberapa teknik encoding dan mengevaluasi dampaknya terhadap performa model. Pemahaman domain sangat penting dalam encoding. Tanpa pemahaman konteks, proses encoding dapat menghasilkan representasi yang menyesatkan. Encoding data kategorikal juga berkaitan dengan interpretabilitas model. Metode seperti One-Hot Encoding lebih mudah diinterpretasikan dibandingkan teknik encoding berbasis hash. Dalam dataset yang dinamis, kategori baru dapat muncul seiring waktu. Oleh karena itu, strategi encoding harus mampu menangani kategori yang belum pernah muncul sebelumnya. Beberapa library Machine Learning modern menyediakan mekanisme untuk menangani kategori baru secara otomatis, sehingga model tetap stabil saat digunakan di lingkungan produksi. Encoding data kategorikal bukan sekadar proses teknis, tetapi juga proses analitis yang memerlukan pertimbangan statistik, komputasi, dan domain pengetahuan. Kesalahan dalam encoding dapat berdampak besar terhadap akurasi dan generalisasi model. Oleh karena itu, tahap ini harus dirancang dengan cermat. Dengan encoding yang tepat, informasi kategorikal dapat dimanfaatkan secara optimal oleh model Machine Learning, sehingga meningkatkan kualitas prediksi dan pengambilan keputusan.

Contoh Implementasi Encoding Data Kategorikal dengan Python:

a. Membuat Dataset Contoh : Dataset simulasi mahasiswa dengan fitur kategorikal nominal dan ordinal.

```
import pandas as pd
data = pd.DataFrame({
  'nama': ['Andi', 'Budi', 'Citra', 'Dina', 'Eka'],
  'jurusan': ['Informatika', 'Sistem Informasi', 'Informatika', 'Teknik Komputer', 'Sistem Informasi'],
```

	nama	jurusan	jenis_kelamin	pendidikan	lulus
0	Andi	Informatika	L	SMA	1
1	Budi	Sistem Informasi	L	D3	1
2	Citra	Informatika	P	S1	1
3	Dina	Teknik Komputer	P	SMA	0
4	Eka	Sistem Informasi	L	S2	1

```
    'jenis_kelamin': ['L', 'L', 'P', 'P', 'L'],
    'pendidikan': ['SMA', 'D3', 'S1', 'SMA', 'S2'],
    'lulus': [1, 1, 1, 0, 1]
})

    print(data)
```

	pendidikan	pendidikan_le
0	SMA	3
1	D3	0
2	S1	1
3	SMA	3
4	S2	2

b. Label Encoding (Cocok untuk Data Ordinal) : Digunakan untuk data yang memiliki urutan.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['pendidikan_le'] = le.fit_transform(data['pendidikan'])
print(data[['pendidikan', 'pendidikan_le']])
```

c. Ordinal Encoding (Menentukan Urutan Secara Manual) : Lebih tepat untuk data ordinal karena urutan ditentukan berdasarkan pengetahuan domain.

```
from sklearn.preprocessing import OrdinalEncoder

# Menentukan urutan pendidikan
education_order = [['SMA', 'D3', 'S1', 'S2']]
ordinal_encoder =
OrdinalEncoder(categories=education_order)
data['pendidikan_ordinal'] =
ordinal_encoder.fit_transform(data[['pendidikan']])
print(data[['pendidikan', 'pendidikan_ordinal']])
```

d. One-Hot Encoding (Cocok untuk Data Nominal) : Digunakan untuk data tanpa urutan, seperti jurusan dan jenis kelamin.

1. One-Hot Encoding dengan Pandas

```
data_ohe = pd.get_dummies(
  data,
```

```
    nama pendidikan  lulus  pendidikan_le  pendidikan_ordinal  \
0   Andi        SMA      1              3                 0.0
1   Budi         D3      1              0                 1.0
2  Citra         S1      1              1                 2.0
3   Dina        SMA      0              3                 0.0
4    Eka         S2      1              2                 3.0

   jurusan_Sistem Informasi  jurusan_Teknik Komputer  jenis_kelamin_P
0                     False                    False            False
1                      True                    False            False
2                     False                    False             True
3                     False                     True             True
4                      True                    False            False
```

```
  columns=['jurusan', 'jenis_kelamin'],
  drop_first=True
)
print(data_ohe)
```

2. One-Hot Encoding dengan Scikit-learn

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse=False, drop='first')
encoded = ohe.fit_transform(data[['jurusan',
'jenis_kelamin']])
encoded_df = pd.DataFrame(
  encoded,
  columns=ohe.get_feature_names_out(['jurusan',
'jenis_kelamin'])
```

```
  pendidikan  pendidikan_ordinal
0        SMA                0.0
1         D3                1.0
2         S1                2.0
3        SMA                0.0
4         S2                3.0
```

```
    )
print(encoded_df)
```

e. Encoding + Model Klasifikasi : Menggabungkan encoding dengan model klasifikasi sederhana.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X = data_ohe.drop(['nama', 'pendidikan', 'lulus'], axis=1)
```

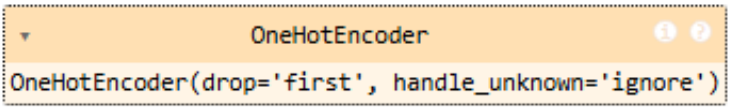
```
Akurasi: 1.0
```

```
y = data_ohe['lulus']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("Akurasi:", accuracy_score(y_test, y_pred))
```

f. Encoding dalam Pipeline (Best Practice) : Pipeline mencegah data leakage dan membuat preprocessing lebih rapi.

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier

# Menentukan kolom
numerical_features = []
categorical_features = ['jurusan', 'jenis_kelamin']

preprocessor = ColumnTransformer(
  transformers=[
    ('cat', OneHotEncoder(drop='first'), categorical_features)
  ],
  remainder='drop'
)

    pipeline = Pipeline(steps=[
```

```
    ('preprocess', preprocessor),
    ('model', RandomForestClassifier(random_state=42))
])

X = data[['jurusan', 'jenis_kelamin']]
y = data['lulus']
```




```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
pipeline.fit(X_train, y_train)
print("Akurasi Pipeline:", pipeline.score(X_test, y_test))
```

g. Menangani Kategori Baru (Unknown Categories) : Sangat penting untuk data produksi.

OneHotEncoder(handle_unknown='ignore', drop='first')

2.4 Normalisasi dan Standardisasi Data

Normalisasi dan standardisasi data merupakan bagian penting dari tahap preprocessing dalam Machine Learning. Kedua teknik ini digunakan untuk menyelaraskan skala fitur agar algoritma pembelajaran mesin dapat bekerja secara optimal. Dalam dataset dunia nyata, fitur sering kali memiliki rentang nilai yang berbeda-beda, sehingga dapat memengaruhi proses pembelajaran model. Banyak algoritma Machine Learning sensitif terhadap perbedaan skala data. Algoritma berbasis jarak, seperti K-Nearest Neighbor (KNN) dan Support Vector Machine (SVM), sangat dipengaruhi oleh besar kecilnya nilai fitur. Tanpa penyelarasan skala, fitur dengan rentang besar akan mendominasi fitur lain, meskipun fitur tersebut tidak lebih penting. Normalisasi dan standardisasi bertujuan untuk menghilangkan bias skala tersebut. Dengan menyamakan skala fitur, setiap variabel memiliki kontribusi yang lebih seimbang terhadap model [10]. Hal ini membantu meningkatkan stabilitas dan kecepatan konvergensi algoritma. Normalisasi adalah teknik penskalaan data ke rentang

tertentu, biasanya antara 0 dan 1. Metode normalisasi yang paling umum adalah Min-Max Normalization. Pada metode ini, setiap nilai fitur ditransformasikan berdasarkan nilai minimum dan maksimum fitur tersebut. Secara matematis, normalisasi Min-Max dirumuskan sebagai:

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

Hasil dari normalisasi ini adalah data yang berada dalam rentang [0, 1]. Teknik ini sangat berguna ketika data tidak mengandung outlier ekstrem dan ketika rentang nilai tertentu diperlukan, misalnya pada Neural Network dengan fungsi aktivasi tertentu. Namun, normalisasi Min-Max memiliki kelemahan utama, yaitu sangat sensitif terhadap outlier. Jika terdapat nilai ekstrem, rentang data akan melebar sehingga sebagian besar data terkompresi ke rentang yang sempit. Selain Min-Max, terdapat juga normalisasi berbasis skala maksimum (*Max Absolute Scaling*), di mana setiap nilai dibagi dengan nilai absolut maksimum fitur.

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

Metode ini mempertahankan tanda positif dan negatif data. Berbeda dengan normalisasi, standardisasi bertujuan untuk mengubah data sehingga memiliki rata-rata (mean) nol dan standar deviasi satu. Teknik ini dikenal sebagai Z-score Standardization. Secara matematis, standardisasi dirumuskan sebagai:

di mana μ adalah mean dan σadalah standar deviasi. Hasil standardisasi tidak dibatasi pada rentang tertentu, tetapi mengikuti distribusi dengan pusat di nol. Standardisasi sangat efektif ketika data berdistribusi mendekati normal (Gaussian). Banyak algoritma, seperti Logistic Regression, SVM, dan Principal Component Analysis (PCA), bekerja lebih baik dengan data yang telah distandardisasi.

Keunggulan utama standardisasi adalah ketahanannya terhadap perbedaan skala antar fitur. Namun, standardisasi juga tetap sensitif terhadap outlier, meskipun tidak separah normalisasi Min-Max. Untuk mengatasi pengaruh outlier, tersedia metode Robust Scaling, yang menggunakan median dan interquartile range (IQR) sebagai dasar transformasi. Metode ini lebih tahan terhadap nilai ekstrem. Pemilihan antara normalisasi dan standardisasi bergantung pada jenis algoritma yang digunakan. Algoritma berbasis jarak dan gradien umumnya memerlukan data yang telah diskalakan, sedangkan algoritma berbasis pohon keputusan relatif tidak sensitif terhadap skala data. Decision Tree, Random Forest, dan Gradient Boosting tidak memerlukan normalisasi atau standardisasi karena proses pemisahan node tidak bergantung pada jarak atau magnitudo nilai. Namun, meskipun tidak wajib, penerapan scaling tetap dapat membantu dalam konsistensi pipeline, terutama jika beberapa algoritma digunakan secara bersamaan dalam perbandingan model.

Normalisasi dan standardisasi juga berdampak pada proses optimasi. Algoritma berbasis gradient descent akan lebih cepat konvergen jika fitur memiliki skala yang seragam. Dalam praktik, normalisasi dan standardisasi harus dilakukan setelah pemisahan data training dan testing. Skala harus dihitung hanya dari data training untuk mencegah data leakage. Kesalahan umum yang sering terjadi adalah melakukan scaling sebelum data dibagi. Hal ini menyebabkan informasi dari data testing bocor ke data training, sehingga evaluasi model menjadi tidak valid. Tools seperti Scikit-learn menyediakan berbagai scaler yang mudah digunakan dan terintegrasi dengan pipeline. Beberapa scaler umum antara lain MinMaxScaler, StandardScaler, dan RobustScaler. Penggunaan pipeline sangat disarankan untuk memastikan proses scaling diterapkan secara konsisten pada data training dan testing. Normalisasi dan standardisasi juga penting dalam konteks feature engineering. Fitur yang telah diskalakan lebih mudah dikombinasikan dan dianalisis secara matematis. Kesalahan dalam pemilihan metode scaling dapat menyebabkan model tidak stabil atau performanya menurun. Oleh karena itu, pemahaman

mendalam mengenai perbedaan normalisasi dan standardisasi menjadi keharusan bagi praktisi Machine Learning.

Contoh Implementasi Normalisasi & Standardisasi Data dengan Python:

a. Membuat Dataset Contoh

```
import pandas as pd
import numpy as np

# Dataset simulasi
data = pd.DataFrame({
   'nilai': [60, 70, 80, 90, 100],
   'kehadiran': [65, 70, 85, 90, 95],
   'jam_belajar': [5, 10, 15, 20, 50]  # mengandung outlier
})

print(data)
```

b. Normalisasi (Min-Max Scaling) : Normalisasi mengubah data ke rentang 0 – 1.

```
from sklearn.preprocessing import MinMaxScaler

scaler_minmax = MinMaxScaler()

data_minmax = pd.DataFrame(
```

```
Hasil Min-Max Normalization:
   nilai  kehadiran  jam_belajar
0   0.00   0.000000     0.000000
1   0.25   0.166667     0.111111
2   0.50   0.666667     0.222222
3   0.75   0.833333     0.333333
4   1.00   1.000000     1.000000
```

```
   scaler_minmax.fit_transform(data),
   columns=data.columns
)

print("Hasil Min-Max Normalization:")
print(data_minmax)
```

c. Standardisasi (Z-score / StandardScaler) : Standardisasi membuat data memiliki mean = 0 dan std = 1.

```
from sklearn.preprocessing import StandardScaler

scaler_standard = StandardScaler()

data_standard = pd.DataFrame(
  scaler_standard.fit_transform(data),
```

	nilai	kehadiran	jam_belajar
0	60	65	5
1	70	70	10
2	80	85	15
3	90	90	20
4	100	95	50

```
  columns=data.columns
)

print("Hasil Standardisasi (Z-score):")
print(data_standard)
```

d. Robust Scaling (Tahan terhadap Outlier) : Menggunakan median dan IQR, lebih tahan terhadap nilai ekstrem.

```
from sklearn.preprocessing import RobustScaler

scaler_robust = RobustScaler()

data_robust = pd.DataFrame(
  scaler_robust.fit_transform(data),
  columns=data.columns
)

print("Hasil Robust Scaling:")
print(data_robust)
```

e. Perbandingan Singkat Hasil Scaling

```
comparison = pd.concat(
```

```
    [data, data_minmax, data_standard, data_robust],
    axis=1,
```

```
Hasil Standardisasi (Z-score):
      nilai  kehadiran  jam_belajar
0 -1.414214  -1.382189    -0.948683
1 -0.707107  -0.950255    -0.632456
2  0.000000   0.345547    -0.316228
3  0.707107   0.777482     0.000000
4  1.414214   1.209416     1.897367
```

```
    keys=['Asli', 'MinMax', 'Standard', 'Robust']
)

print(comparison)
```

f. Scaling + Train-Test Split (Best Practice) : Jangan melakukan scaling sebelum split data!

```
from sklearn.model_selection import train_test_split

X = data
y = [0, 0, 1, 1, 1]  # label contoh

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # fit hanya di train
X_test_scaled = scaler.transform(X_test)       # transform test

print("Train Scaled:\n", X_train_scaled)
    print("Test Scaled:\n", X_test_scaled)
```

```
Train Scaled:
 [[ 0.26726124  0.46291005  0.26726124]
 [-1.33630621 -1.38873015 -1.33630621]
 [ 1.06904497  0.9258201   1.06904497]]
Test Scaled:
 [[-0.53452248 -0.9258201  -0.53452248]
 [ 1.87082869  1.38873015  5.87974732]]
```

g. Scaling dalam Pipeline : Pipeline memastikan tidak ada data leakage.

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
```

```
   Asli                       MinMax                        Standard
  nilai kehadiran jam_belajar  nilai kehadiran jam_belajar     nilai
0    60        65           5   0.00  0.000000    0.000000 -1.414214
1    70        70          10   0.25  0.166667    0.111111 -0.707107
2    80        85          15   0.50  0.666667    0.222222  0.000000
3    90        90          20   0.75  0.833333    0.333333  0.707107
4   100        95          50   1.00  1.000000    1.000000  1.414214

                        Robust
  kehadiran jam_belajar  nilai kehadiran jam_belajar
0 -1.382189   -0.948683   -1.0     -1.00        -1.0
1 -0.950255   -0.632456   -0.5     -0.75        -0.5
2  0.345547   -0.316228    0.0      0.00         0.0
3  0.777482    0.000000    0.5      0.25         0.5
4  1.209416    1.897367    1.0      0.50         3.5
```

```
from sklearn.metrics import accuracy_score
pipeline = Pipeline(steps=[
   ('scaler', StandardScaler()),
   ('model', LogisticRegression())
])
pipeline.fit(X_train, y_train)
print("Akurasi Model:", pipeline.score(X_test, y_test))
```

Membandingkan Performa Model Sebelum vs Sesudah Scaling

a. Membuat Dataset

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

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Set seed
```

```
np.random.seed(42)

# Fitur 1 (skala kecil, sangat informatif)
feature1 = np.random.rand(200)

# Fitur 2 (skala besar, tidak informatif)
feature2 = np.random.rand(200) * 10000

# Target berdasarkan feature1
y = (feature1 > 0.5).astype(int)

# Gabungkan
X = np.column_stack((feature1, feature2))
```

```
Akurasi Model: 1.0
```

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

b. KNN Tanpa Scaling

```
knn_no_scaling = KNeighborsClassifier(n_neighbors=5)
knn_no_scaling.fit(X_train, y_train)

y_pred_no = knn_no_scaling.predict(X_test)
acc_no = accuracy_score(y_test, y_pred_no)
```

```
Akurasi KNN Tanpa Scaling: 0.48333333333333334
```

```
print("Akurasi KNN Tanpa Scaling:", acc_no)
```

c. KNN dengan Scaling

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
```

```
Akurasi KNN Dengan Scaling: 0.9666666666666667
```

```
X_test_scaled = scaler.transform(X_test)

knn_scaled = KNeighborsClassifier(n_neighbors=5)
knn_scaled.fit(X_train_scaled, y_train)

y_pred_scaled = knn_scaled.predict(X_test_scaled)
acc_scaled = accuracy_score(y_test, y_pred_scaled)

print("Akurasi KNN Dengan Scaling:", acc_scaled)
```

d. Perbandingan Hasil

```
print("\nPerbandingan:")
print("Tanpa Scaling :", acc_no)
print("Dengan Scaling:", acc_scaled)
```

Hasil perbandingan menunjukkan perbedaan performa yang sangat signifikan antara model yang dilatih tanpa scaling dan model yang dilatih dengan scaling. Tanpa proses scaling, akurasi model hanya mencapai sekitar 48,33%, yang mengindikasikan bahwa model hampir tidak mampu mengenali pola klasifikasi secara efektif dan kinerjanya mendekati tebakan acak. Kondisi ini terjadi karena adanya perbedaan skala fitur yang sangat ekstrem, sehingga fitur dengan nilai numerik besar mendominasi perhitungan jarak dan menyebabkan fitur yang sebenarnya lebih informatif menjadi terabaikan oleh model. Sebaliknya, setelah dilakukan proses scaling, akurasi model meningkat secara drastis hingga sekitar 96,67%. Peningkatan ini menunjukkan bahwa standardisasi berhasil menyeimbangkan kontribusi setiap fitur dalam proses pembelajaran. Dengan skala yang seragam, algoritma klasifikasi berbasis jarak dapat menghitung kedekatan antar data secara lebih representatif, sehingga pola yang relevan dapat dipelajari dengan baik. Hasil ini menegaskan bahwa scaling bukan sekadar langkah tambahan, melainkan tahapan preprocessing yang krusial untuk

memastikan model mampu belajar dari data secara optimal dan menghasilkan performa yang jauh lebih akurat.

```
Perbandingan:
Tanpa Scaling : 0.48333333333333334
Dengan Scaling: 0.9666666666666667
```

2.5 Feature Selection dan Feature Engineering

Feature selection dan feature engineering merupakan dua tahap penting dalam preprocessing data yang bertujuan untuk meningkatkan kualitas representasi data sebelum digunakan dalam pemodelan Machine Learning. Kedua proses ini berfokus pada bagaimana fitur dipilih, dimodifikasi, atau dibentuk agar informasi yang relevan dapat dimanfaatkan secara optimal oleh algoritma pembelajaran mesin. Feature selection adalah proses memilih subset fitur yang paling relevan dari sekumpulan fitur yang tersedia. Tujuan utama dari feature selection adalah menghilangkan fitur yang tidak relevan, redundan, atau bersifat noise sehingga model dapat belajar lebih efektif. Dengan jumlah fitur yang lebih sedikit namun informatif, model menjadi lebih sederhana dan lebih mudah digeneralisasi. Salah satu alasan utama dilakukan feature selection adalah untuk mengurangi kompleksitas model. Dataset dunia nyata sering kali memiliki jumlah fitur yang sangat besar, bahkan ratusan atau ribuan. Tidak semua fitur tersebut berkontribusi secara signifikan terhadap target, dan keberadaan fitur yang tidak relevan dapat menurunkan performa model. Feature selection juga berperan penting dalam mengatasi masalah curse of dimensionality. Ketika jumlah fitur meningkat, ruang fitur menjadi sangat besar sehingga jarak antar data menjadi kurang bermakna, khususnya pada algoritma berbasis jarak seperti KNN [11]. Dengan mengurangi dimensi fitur, model dapat bekerja lebih stabil dan efisien. Selain meningkatkan performa, feature selection juga meningkatkan interpretabilitas model. Model dengan fitur yang lebih sedikit lebih mudah dipahami dan dianalisis, terutama dalam konteks pengambilan keputusan di bidang kesehatan, keuangan, atau kebijakan publik.

Metode feature selection secara umum dapat dikategorikan menjadi tiga pendekatan utama, yaitu [12]:

a. Filter Methods

Filter methods merupakan pendekatan feature selection yang mengevaluasi fitur berdasarkan karakteristik statistik data, tanpa melibatkan algoritma Machine Learning tertentu. Metode ini menilai hubungan antara fitur dan variabel target secara independen, sehingga bersifat model-agnostic. Keunggulan utama filter methods adalah efisiensi komputasi. Karena tidak melibatkan proses pelatihan model, metode ini sangat cepat dan cocok digunakan pada dataset dengan jumlah fitur yang sangat besar, seperti data teks atau data genomik. Filter methods bekerja dengan mengukur kekuatan hubungan antara setiap fitur dengan target. Fitur yang memiliki hubungan kuat dianggap relevan, sedangkan fitur dengan hubungan lemah atau tidak signifikan dapat dihilangkan. Namun, metode ini tidak mempertimbangkan interaksi antar fitur. Keterbatasan filter methods terletak pada sifatnya yang univariat. Fitur yang secara individual tampak tidak penting bisa saja menjadi sangat informatif ketika dikombinasikan dengan fitur lain. Oleh karena itu, filter methods sering digunakan sebagai tahap awal sebelum metode seleksi yang lebih kompleks.

Contoh Filter Methods:

- Korelasi Pearson
- Chi-Square Test
- Mutual Information
- ANOVA F-test

Contoh Implementasi (Chi-Square):

```
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
selector = SelectKBest(score_func=chi2, k=2)
X_selected = selector.fit_transform(X, y)
print("Jumlah fitur setelah seleksi:", X_selected.shape[1])
```

b. Wrapper Methods

Wrapper methods merupakan pendekatan feature selection yang menggunakan algoritma Machine Learning sebagai evaluator. Metode ini mencari subset fitur terbaik berdasarkan performa model, seperti akurasi atau error. Pendekatan ini bekerja dengan cara mencoba berbagai kombinasi fitur, melatih model pada setiap kombinasi, lalu memilih subset fitur yang menghasilkan performa terbaik. Karena melibatkan proses training berulang, wrapper methods cenderung lebih akurat dibandingkan filter methods. Keunggulan utama wrapper methods adalah kemampuannya dalam mempertimbangkan interaksi antar fitur. Metode ini dapat mengidentifikasi kombinasi fitur yang secara kolektif memberikan kontribusi besar terhadap performa model. Namun, wrapper methods memiliki kelemahan utama berupa biaya komputasi yang tinggi, terutama pada dataset dengan jumlah fitur besar. Oleh karena itu, metode ini lebih cocok untuk dataset dengan jumlah fitur terbatas.

Contoh Wrapper Methods:

- Forward Selection
- Backward Elimination
- Recursive Feature Elimination (RFE)

Contoh Implementasi (RFE)

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
model = LogisticRegression(max_iter=200)
rfe = RFE(estimator=model, n_features_to_select=2)
X_selected = rfe.fit_transform(X, y)
print("Jumlah fitur setelah RFE:", X_selected.shape[1])
```

c. Embedded Methods

Embedded methods merupakan pendekatan feature selection yang terintegrasi langsung dalam proses pelatihan model. Seleksi fitur dilakukan secara otomatis saat model

dilatih, sehingga lebih efisien dibandingkan wrapper methods. Pendekatan ini memanfaatkan mekanisme internal model untuk menentukan fitur penting. Contohnya, regularisasi L1 (Lasso) akan menekan koefisien fitur yang tidak relevan hingga mendekati nol. Keunggulan embedded methods adalah keseimbangan antara efisiensi dan akurasi. Metode ini mempertimbangkan hubungan fitur dengan target sekaligus interaksi antar fitur, tanpa memerlukan evaluasi subset fitur secara eksplisit. Namun, embedded methods bersifat model dependent. Artinya, fitur yang dipilih sangat bergantung pada algoritma yang digunakan, sehingga hasil seleksi dapat berbeda jika model diganti.

Contoh Embedded Methods:

- Lasso (L1 Regularization)
- Elastic Net
- Feature Importance pada Decision Tree / Random Forest

Contoh Implementasi (Random Forest Feature Importance)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
import numpy as np
```

```
Indeks fitur terpenting: [2 3]
```

```
X, y = load_iris(return_X_y=True)

rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)
importances = rf.feature_importances_
important_features = np.argsort(importances)[::-1][:2]
print("Indeks fitur terpenting:", important_features)
```

2.6 Pembagian Data Training dan Testing

Pembagian data training dan testing merupakan salah satu tahapan fundamental dalam proyek Machine Learning yang bertujuan untuk mengevaluasi kemampuan generalisasi model.

Generalisasi mengacu pada kemampuan model untuk memberikan prediksi yang baik pada data baru yang belum pernah dilihat sebelumnya. Tanpa pembagian data yang tepat, evaluasi model dapat menjadi bias dan tidak mencerminkan performa sebenarnya. Data training adalah bagian dari dataset yang digunakan untuk melatih model Machine Learning. Pada tahap ini, algoritma belajar mengenali pola, hubungan, dan struktur dari data berdasarkan fitur dan label yang tersedia. Proses pelatihan bertujuan untuk menemukan parameter model yang meminimalkan kesalahan prediksi pada data training. Sebaliknya, data testing digunakan untuk menguji performa model setelah proses pelatihan selesai [13]. Data ini bersifat independen dari data training dan tidak boleh digunakan dalam proses pembelajaran model. Evaluasi pada data testing memberikan gambaran seberapa baik model dapat bekerja pada data yang benar-benar baru. Pembagian data training dan testing dilakukan untuk menghindari overfitting, yaitu kondisi ketika model terlalu menyesuaikan diri dengan data training sehingga performanya menurun saat dihadapkan pada data baru. Dengan adanya data testing, overfitting dapat terdeteksi secara objektif. Secara umum, proporsi pembagian data training dan testing yang sering digunakan adalah 70:30, 80:20, atau 90:10. Pemilihan proporsi bergantung pada ukuran dataset dan kompleksitas model. Dataset yang besar memungkinkan porsi data training lebih besar, sedangkan dataset kecil memerlukan pertimbangan khusus. Pada dataset dengan ukuran terbatas, pembagian data yang tidak tepat dapat menyebabkan data training atau testing menjadi terlalu sedikit. Hal ini dapat menghasilkan estimasi performa yang tidak stabil. Dalam kasus seperti ini, teknik validasi silang sering digunakan sebagai alternatif.

Pembagian data harus dilakukan secara acak (*random split*) agar distribusi data training dan testing serupa. Pembagian yang tidak acak dapat menyebabkan bias, terutama jika data memiliki pola tertentu seperti urutan waktu atau kelompok tertentu. Namun, pada data berbasis waktu (*time series*), pembagian acak tidak dianjurkan. Data training harus berasal dari periode waktu sebelumnya, sedangkan data testing berasal dari periode

setelahnya. Hal ini bertujuan untuk mensimulasikan kondisi prediksi di dunia nyata. Dalam permasalahan klasifikasi, penting untuk memastikan bahwa distribusi kelas pada data training dan testing relatif seimbang. Jika distribusi kelas berbeda secara signifikan, evaluasi model dapat menjadi menyesatkan. Untuk mengatasi masalah ini, digunakan teknik stratified split, yaitu pembagian data yang mempertahankan proporsi kelas yang sama pada data training dan testing. Teknik ini sangat penting pada dataset yang tidak seimbang. Pembagian data training dan testing juga berhubungan erat dengan masalah data leakage [14]. Data leakage terjadi ketika informasi dari data testing secara tidak sengaja digunakan dalam proses training. Hal ini dapat menyebabkan performa model terlihat sangat baik, padahal tidak realistis. Contoh data leakage yang umum adalah melakukan normalisasi atau encoding sebelum data dibagi menjadi training dan testing. Skala atau kategori yang dihitung dari seluruh dataset akan bocor ke data training, sehingga evaluasi menjadi tidak valid. Oleh karena itu, pembagian data harus dilakukan sebelum tahapan preprocessing seperti scaling, encoding, dan feature selection. Preprocessing harus dilatih hanya menggunakan data training dan kemudian diterapkan pada data testing. Dalam praktik modern, pembagian data sering diintegrasikan dalam pipeline Machine Learning untuk memastikan konsistensi dan mencegah kesalahan prosedural.

Selain data training dan testing, sering kali digunakan data validation. Data validation digunakan untuk tuning hyperparameter dan pemilihan model, sedangkan data testing disimpan untuk evaluasi akhir. Penggunaan data validation membantu mencegah overfitting pada data training sekaligus menghindari bias pada data testing. Dalam penelitian ilmiah, pemisahan ini sangat dianjurkan. Pada dataset berskala besar, pembagian data dapat dilakukan secara lebih fleksibel. Namun, prinsip independensi antara training dan testing tetap harus dijaga. Pemilihan metode pembagian data juga harus mempertimbangkan tujuan penelitian. Jika fokus pada prediksi jangka panjang, pembagian berbasis waktu lebih tepat. Jika fokus pada klasifikasi

umum, pembagian acak atau stratified lebih sesuai. Dalam evaluasi model, hasil pada data testing sering dianggap sebagai indikator utama performa model. Namun, evaluasi ini hanya valid jika pembagian data dilakukan dengan benar. Kesalahan dalam pembagian data dapat menyebabkan kesimpulan yang salah, seperti menganggap model sangat akurat padahal sebenarnya tidak mampu mengeneralisasi. Oleh karena itu, dokumentasi metode pembagian data menjadi bagian penting dalam laporan penelitian dan publikasi ilmiah. Dokumentasi ini mencakup proporsi pembagian, metode pembagian (acak, stratified, berbasis waktu), serta alasan pemilihan metode tersebut.

Contoh implementasi pembagian data training–testing menggunakan Python:

a. Random Train Test Split : Digunakan ketika data tidak memiliki urutan waktu dan distribusi kelas relatif seimbang.

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

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

# Pembagian data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
```

```
Jumlah data training: 120
Jumlah data testing : 30
```

```
    X, y,
    test_size=0.2,
    random_state=42,
    shuffle=True
)

print("Jumlah data training:", X_train.shape[0])
print("Jumlah data testing :", X_test.shape[0])
```

b. Stratified Train Test Split : Digunakan ketika distribusi kelas tidak seimbang, agar proporsi kelas di training dan testing tetap sama.

```
from sklearn.model_selection import train_test_split
import numpy as np

# Cek distribusi kelas awal
print("Distribusi kelas awal:", np.bincount(y))

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y   # ✓ kunci stratified split
)

print("Distribusi kelas training:", np.bincount(y_train))
print("Distribusi kelas testing :", np.bincount(y_test))
```

c. Train Validation Test Split : Digunakan ketika ingin:
   - Melatih model
   - Tuning hyperparameter
   - Evaluasi akhir yang objektif

```
Langkah 1: Training + Testing
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

Langkah 2: Training + Validation
X_train, X_val, y_train, y_val = train_test_split(
```

```
Training : 90
Validation: 30
Testing  : 30
```

```
    X_temp, y_temp,
    test_size=0.25,   # 0.25 × 0.8 = 0.2
    random_state=42
)

print("Training :", X_train.shape[0])
```

```
Distribusi kelas awal: [50 50 50]
Distribusi kelas training: [40 40 40]
Distribusi kelas testing : [10 10 10]
```

```
print("Validation:", X_val.shape[0])
print("Testing  :", X_test.shape[0])
```

d. Pembagian Data untuk Time Series (Tanpa Shuffle) : JANGAN gunakan random split pada data time series

```
import pandas as pd

# Dataset time series contoh
data = pd.DataFrame({
    'tanggal': pd.date_range(start='2023-01-01', periods=100),
    'nilai': range(100)
```

```
Training data: (80, 2)
Testing data : (20, 2)
```

```
})

# 80% data awal → training
train_size = int(len(data) * 0.8)

train_data = data.iloc[:train_size]
test_data = data.iloc[train_size:]

print("Training data:", train_data.shape)
```

```
Akurasi: 1.0
```

```
print("Testing data :", test_data.shape)
```

e. Pembagian Data + Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Akurasi:", accuracy_score(y_test, y_pred))
```

BAB 3
Metode Klasifikasi 1: Support Vector Machine (SVM)

3.1 Konsep Dasar Support Vector Machine

Support Vector Machine (SVM) merupakan salah satu algoritma Machine Learning yang sangat populer dan kuat, khususnya untuk permasalahan klasifikasi dan regresi. SVM termasuk ke dalam kategori supervised learning, karena proses pembelajarannya memerlukan data berlabel. Keunggulan utama SVM terletak pada kemampuannya membangun batas keputusan (decision boundary) yang optimal dan memiliki kemampuan generalisasi yang baik, bahkan pada dataset berdimensi tinggi. Ide dasar SVM adalah mencari sebuah hyperplane yang dapat memisahkan data dari kelas yang berbeda dengan jarak pemisah (margin) yang maksimal. Berbeda dengan algoritma klasifikasi lain yang hanya berfokus pada kesalahan klasifikasi, SVM secara eksplisit berusaha memaksimalkan jarak antara hyperplane dan titik data terdekat dari masing-masing kelas. Pendekatan ini membuat SVM lebih robust terhadap noise dan overfitting [15].

Dalam kasus dua dimensi, hyperplane dapat berupa sebuah garis lurus. Pada tiga dimensi, hyperplane menjadi sebuah bidang. Sedangkan pada dimensi yang lebih tinggi, hyperplane merupakan objek geometris berdimensi lebih dari tiga yang secara matematis tetap dapat didefinisikan. Dengan demikian, SVM secara alami dapat menangani data dengan banyak fitur. Titik-titik data yang paling dekat dengan hyperplane disebut support vectors. Support vectors memiliki peran yang sangat penting karena hanya titik-titik inilah yang menentukan posisi dan orientasi hyperplane. Data lain yang berada lebih jauh dari hyperplane tidak memengaruhi hasil model secara langsung. Inilah alasan mengapa algoritma ini dinamakan *Support Vector Machine*. Secara matematis, tujuan SVM adalah menemukan vektor bobot w dan bias b yang memaksimalkan margin, dengan tetap memastikan bahwa seluruh data diklasifikasikan dengan benar atau dengan kesalahan

seminimal mungkin. Margin didefinisikan sebagai jarak antara hyperplane dan support vectors dari masing-masing kelas. Untuk kasus klasifikasi linear yang dapat dipisahkan secara sempurna (*linearly separable*), SVM disebut sebagai Hard Margin SVM. Pada pendekatan ini, tidak ada toleransi kesalahan sama sekali. Semua data harus berada di sisi yang benar dari hyperplane [16]. Meskipun ideal secara teori, hard margin jarang digunakan dalam praktik karena data dunia nyata hampir selalu mengandung noise. Untuk mengatasi kondisi tersebut, diperkenalkan Soft Margin SVM, yang memungkinkan beberapa data berada di sisi yang salah dari hyperplane. Soft margin menggunakan variabel slack untuk mengukur tingkat pelanggaran terhadap margin. Pendekatan ini memberikan keseimbangan antara margin yang lebar dan jumlah kesalahan klasifikasi.

Parameter penting dalam soft margin SVM adalah parameter C. Parameter C mengontrol tingkat penalti terhadap kesalahan klasifikasi. Nilai C yang besar akan menghukum kesalahan dengan keras sehingga model cenderung overfitting, sedangkan nilai C yang kecil memberikan toleransi lebih besar terhadap kesalahan sehingga model lebih sederhana dan memiliki generalisasi yang lebih baik. SVM tidak hanya terbatas pada klasifikasi linear. Banyak permasalahan nyata tidak dapat dipisahkan secara linear. Untuk mengatasi hal ini, SVM menggunakan konsep kernel trick. Kernel trick memungkinkan data dipetakan ke ruang berdimensi lebih tinggi, di mana pemisahan linear menjadi mungkin, tanpa harus menghitung transformasi tersebut secara eksplisit. Beberapa fungsi kernel yang umum digunakan antara lain Linear Kernel, Polynomial Kernel, Radial Basis Function (RBF) Kernel, dan Sigmoid Kernel. Masing-masing kernel memiliki karakteristik yang berbeda dan cocok untuk jenis data tertentu. RBF kernel merupakan salah satu yang paling sering digunakan karena fleksibilitasnya dalam menangkap pola non-linear. Kernel RBF bekerja dengan mengukur jarak antar data dalam ruang fitur dan memetakan data ke ruang berdimensi tak hingga. Parameter penting pada kernel RBF adalah gamma, yang mengontrol seberapa besar pengaruh satu titik data terhadap

titik lain. Nilai gamma besar membuat model sangat sensitif terhadap data, sedangkan nilai gamma kecil menghasilkan batas keputusan yang lebih halus. Selain untuk klasifikasi, SVM juga dapat digunakan untuk regresi melalui pendekatan Support Vector Regression (SVR). Pada SVR, tujuan bukan memisahkan kelas, melainkan mencari fungsi regresi yang memiliki deviasi maksimum tertentu terhadap data, yang disebut sebagai epsilon-insensitive loss. SVM memiliki keunggulan utama dalam menangani data berdimensi tinggi dan dataset dengan jumlah fitur lebih banyak dibandingkan jumlah sampel. Hal ini menjadikan SVM sangat populer dalam aplikasi seperti bioinformatika, pengolahan teks, dan deteksi anomali. Namun demikian, SVM juga memiliki keterbatasan. Proses pelatihan SVM dapat menjadi sangat lambat pada dataset yang sangat besar, karena kompleksitas komputasinya bergantung pada jumlah data. Selain itu, pemilihan kernel dan parameter yang tepat membutuhkan eksperimen dan pemahaman yang baik terhadap data.

Dalam praktik, performa SVM sangat dipengaruhi oleh proses preprocessing data, terutama normalisasi atau standardisasi. Karena SVM berbasis perhitungan jarak, perbedaan skala fitur dapat berdampak signifikan terhadap hasil model. Oleh karena itu, scaling data hampir selalu menjadi langkah wajib sebelum menerapkan SVM. SVM juga dikenal memiliki kemampuan generalisasi yang kuat karena prinsip Structural Risk Minimization, yang berfokus pada keseimbangan antara kompleksitas model dan kesalahan pelatihan. Prinsip ini berbeda dengan pendekatan Empirical Risk Minimization yang hanya meminimalkan kesalahan pada data training. Dalam konteks klasifikasi modern, SVM sering digunakan sebagai baseline yang kuat. Meskipun saat ini banyak model kompleks seperti ensemble dan deep learning, SVM tetap relevan karena stabil, teoritis kuat, dan efektif pada banyak jenis permasalahan.

Perbandingan Hard Margin vs Soft Margin Support Vector Machine

Hard Margin dan Soft Margin merupakan dua pendekatan utama dalam Support Vector Machine yang membedakan bagaimana model memperlakukan kesalahan klasifikasi dan noise pada data. Perbedaan keduanya terletak pada tingkat toleransi terhadap pelanggaran margin serta asumsi yang digunakan terhadap struktur data. Pemahaman perbedaan ini sangat penting karena menentukan apakah SVM dapat bekerja secara realistis pada data dunia nyata [17].

a. Hard Margin SVM

Hard Margin SVM adalah bentuk paling dasar dari SVM yang mengasumsikan bahwa data dapat dipisahkan secara sempurna oleh sebuah hyperplane linear. Pada pendekatan ini, tidak ada satu pun data yang diperbolehkan berada di sisi yang salah dari hyperplane atau berada di dalam margin. Semua data harus diklasifikasikan dengan benar. Secara konseptual, Hard Margin SVM berusaha memaksimalkan margin dengan batasan yang sangat ketat. Margin dibentuk oleh support vectors yang berada tepat di batas margin, dan semua titik data lainnya harus berada di luar margin tersebut. Pendekatan ini menghasilkan hyperplane yang sangat "tegas" dan presisi. Keunggulan utama Hard Margin SVM adalah kesederhanaan dan kejelasan geometrisnya. Jika data benar-benar bersih, bebas noise, dan linearly separable, Hard Margin SVM dapat menghasilkan model dengan generalisasi yang sangat baik. Dalam kondisi ideal, margin maksimum yang dihasilkan memberikan pemisahan kelas yang optimal. Namun, kelemahan Hard Margin SVM sangat signifikan dalam praktik. Data dunia nyata hampir selalu mengandung noise, outlier, atau overlap antar kelas. Kehadiran satu saja outlier dapat menyebabkan Hard Margin SVM gagal menemukan hyperplane yang valid, sehingga model menjadi tidak stabil dan tidak dapat digunakan.

b. Soft Margin SVM

Soft Margin SVM dikembangkan untuk mengatasi keterbatasan Hard Margin dengan memperkenalkan toleransi terhadap kesalahan klasifikasi. Pada pendekatan ini, beberapa data diperbolehkan berada di dalam margin atau bahkan berada di sisi yang salah dari hyperplane. Pelanggaran tersebut diukur menggunakan variabel slack. Soft Margin SVM tidak hanya berfokus pada pemisahan data secara sempurna, tetapi juga mempertimbangkan keseimbangan antara lebar margin dan jumlah kesalahan klasifikasi. Dengan demikian, model menjadi lebih fleksibel dan realistis ketika diterapkan pada data yang kompleks dan noisy. Parameter kunci dalam Soft Margin SVM adalah parameter C, yang mengontrol tingkat penalti terhadap kesalahan klasifikasi. Nilai C yang besar akan menghukum kesalahan secara ketat, sehingga model mendekati perilaku Hard Margin. Sebaliknya, nilai C yang kecil memberikan toleransi lebih besar terhadap kesalahan, sehingga margin menjadi lebih lebar. Keunggulan utama Soft Margin SVM adalah kemampuannya menangani noise dan outlier dengan lebih baik. Model menjadi lebih robust dan memiliki kemampuan generalisasi yang lebih tinggi dibandingkan Hard Margin, terutama pada dataset dunia nyata yang jarang bersih sempurna. Namun, Soft Margin SVM juga memiliki tantangan, yaitu pemilihan nilai C yang tepat. Nilai C yang terlalu besar dapat menyebabkan overfitting karena model terlalu menyesuaikan diri dengan data training. Sebaliknya, nilai C yang terlalu kecil dapat menyebabkan underfitting karena model terlalu permisif terhadap kesalahan.

3.2 Hyperplane dan Margin

Dalam Support Vector Machine (SVM), konsep hyperplane dan margin merupakan inti dari mekanisme pembelajaran model. Kedua konsep ini menentukan bagaimana data dipisahkan dan seberapa baik model dapat melakukan generalisasi terhadap data yang belum pernah dilihat sebelumnya. Berbeda dengan algoritma klasifikasi lain yang hanya mencari batas pemisah apa pun, SVM

secara eksplisit mencari batas pemisah yang optimal berdasarkan prinsip geometris. Hyperplane dapat didefinisikan sebagai suatu batas keputusan yang memisahkan ruang fitur menjadi dua bagian, masing-masing mewakili kelas yang berbeda. Secara matematis, hyperplane dinyatakan dalam bentuk persamaan linear $w \cdot x + b = 0$, di mana $\mathbf{w}$ adalah vektor bobot yang menentukan orientasi hyperplane, $\mathbf{x}$ adalah vektor fitur, dan b adalah bias yang menentukan posisi hyperplane terhadap titik asal. Dalam ruang dua dimensi, hyperplane berupa sebuah garis lurus. Pada ruang tiga dimensi, hyperplane berbentuk bidang datar. Sedangkan pada ruang berdimensi lebih tinggi, hyperplane merupakan objek geometris abstrak yang tetap dapat direpresentasikan secara matematis. Dengan demikian, konsep hyperplane bersifat umum dan dapat diterapkan pada data dengan jumlah fitur berapa pun.

Tujuan utama SVM adalah mencari hyperplane yang mampu memisahkan data dari kelas yang berbeda secara optimal. Optimalitas di sini tidak hanya berarti memisahkan data dengan benar, tetapi juga memaksimalkan jarak antara hyperplane dan titik data terdekat dari masing-masing kelas. Jarak inilah yang disebut sebagai margin. Margin didefinisikan sebagai jarak minimum antara hyperplane dan support vectors, yaitu titik-titik data yang paling dekat dengan hyperplane dari setiap kelas. Margin mencerminkan tingkat kepercayaan model dalam memisahkan kelas. Semakin besar margin, semakin jauh hyperplane dari data, sehingga model cenderung lebih robust terhadap noise dan variasi data. Secara geometris, terdapat dua hyperplane pendukung yang sejajar dengan hyperplane utama, masing-masing berada pada sisi kelas positif dan negatif. Hyperplane pendukung ini melewati support vectors dan membentuk batas margin. Jarak antara dua hyperplane pendukung inilah yang disebut sebagai margin maksimum. SVM berusaha memaksimalkan margin dengan meminimalkan norma vektor bobot $\|w\|$. Dengan meminimalkan norma bobot, hyperplane akan menjadi lebih "rata" dan jarak ke data terdekat akan semakin besar. Proses ini merupakan inti dari formulasi optimasi SVM. Dalam kasus hard margin SVM, diasumsikan bahwa data dapat

dipisahkan secara sempurna oleh sebuah hyperplane tanpa adanya kesalahan klasifikasi. Semua titik data harus berada di luar margin dan pada sisi yang benar dari hyperplane. Meskipun ideal secara teori, pendekatan ini sangat sensitif terhadap noise dan jarang digunakan dalam data dunia nyata. Untuk mengatasi keterbatasan hard margin, diperkenalkan konsep soft margin SVM. Pada pendekatan ini, beberapa data diperbolehkan berada di dalam margin atau bahkan berada di sisi yang salah dari hyperplane. Pelanggaran ini diukur menggunakan variabel slack, yang memungkinkan fleksibilitas dalam pembentukan margin. Dalam soft margin SVM, terdapat trade-off antara memaksimalkan margin dan meminimalkan kesalahan klasifikasi. Trade-off ini dikontrol oleh parameter C [18]. Nilai C yang besar akan memaksa model untuk mengklasifikasikan data dengan benar meskipun margin menjadi sempit, sedangkan nilai C yang kecil akan memperlebar margin dengan mengizinkan lebih banyak kesalahan.

Pemilihan nilai C sangat memengaruhi posisi hyperplane dan lebar margin. Nilai C yang terlalu besar dapat menyebabkan overfitting karena hyperplane terlalu menyesuaikan diri dengan data training. Sebaliknya, nilai C yang terlalu kecil dapat menyebabkan underfitting karena hyperplane menjadi terlalu sederhana. Konsep hyperplane dan margin juga berlaku pada SVM non-linear melalui penggunaan kernel. Meskipun hyperplane dibentuk di ruang fitur berdimensi tinggi, prinsip margin maksimum tetap dipertahankan. Dengan kernel trick, SVM dapat membangun batas keputusan non-linear yang tetap optimal secara geometris. Keunggulan utama pendekatan margin maksimum adalah kemampuannya dalam meningkatkan generalisasi. Dengan margin yang besar, model tidak terlalu sensitif terhadap variasi kecil pada data input. Hal ini membuat SVM sering menghasilkan performa yang stabil pada data baru. Namun, perlu diperhatikan bahwa margin yang terlalu besar tidak selalu menghasilkan performa terbaik. Jika margin terlalu lebar dan banyak data diabaikan, model dapat kehilangan informasi penting. Oleh karena itu, keseimbangan antara margin dan kesalahan klasifikasi menjadi

kunci keberhasilan SVM. Pemahaman konsep hyperplane dan margin juga penting dalam interpretasi hasil SVM. Support vectors dapat dianggap sebagai contoh data yang paling kritis karena perubahan kecil pada titik-titik ini dapat mengubah posisi hyperplane secara signifikan.

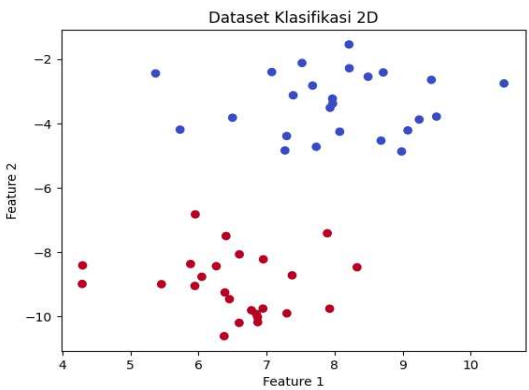
Berikut visualisasi hyperplane dan margin pada Support Vector Machine (SVM) menggunakan Python:

a. Import Library dan Dataset Contoh

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_blobs
```

b. Membuat Dataset Klasifikasi Linear (2 Kelas)

```
# Dataset sintetis 2 dimensi
X, y = make_blobs(
    n_samples=50,
    centers=2,
    random_state=6,
    cluster_std=1.0
)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Dataset Klasifikasi 2D")
plt.show()
```



c. Melatih Model SVM Linear

```
# Model SVM Linear
svm = SVC(kernel='linear', C=1.0)
svm.fit(X, y)
```

d. Mengambil Parameter Hyperplane : Hyperplane SVM dinyatakan sebagai

```
w = svm.coef_[0]   # bobot
b = svm.intercept_[0]  # bias

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

```
Bobot (w): [-0.25395113 -0.83807751]
Bias (b): -3.211559897362612
```

```
print("Bias (b):", b)
```

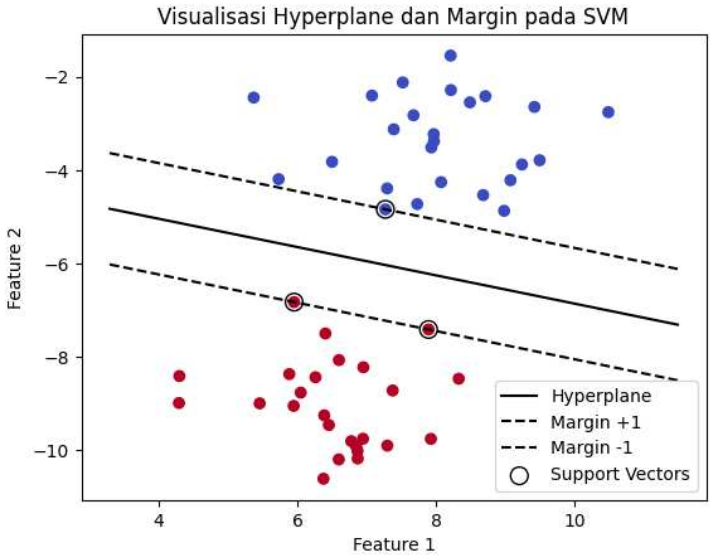
e. Menggambar Hyperplane dan Margin

```
# Membuat grid untuk visualisasi
x_vals = np.linspace(X[:, 0].min() - 1, X[:, 0].max() + 1, 100)

# Persamaan garis
y_hyperplane = -(w[0] * x_vals + b) / w[1]
y_margin_pos = -(w[0] * x_vals + b - 1) / w[1]
y_margin_neg = -(w[0] * x_vals + b + 1) / w[1]
```

$$w_1 x_1 + w_2 x_2 + b = 0$$

```
# Plot data
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
# Plot hyperplane dan margin
plt.plot(x_vals, y_hyperplane, 'k-', label='Hyperplane')
plt.plot(x_vals, y_margin_pos, 'k--', label='Margin +1')
plt.plot(x_vals, y_margin_neg, 'k--', label='Margin -1')
# Highlight support vectors
   plt.scatter(
   svm.support_vectors_[:, 0],
   svm.support_vectors_[:, 1],
```

```
    s=100,
       facecolors='none',
       edgecolors='k',
            label='Support Vectors'
        )
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend()
    plt.title("Visualisasi Hyperplane dan Margin pada SVM")
    plt.show()
```

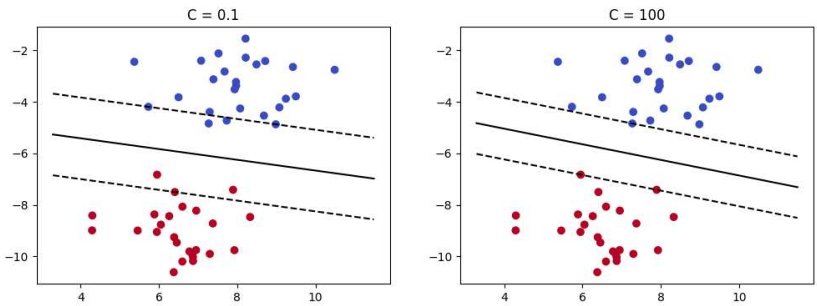


f. Interpretasi Visualisasi
   - Garis tengah (solid) → Hyperplane optimal
   - Dua garis putus-putus → Margin
   - Titik yang dilingkari → Support vectors
   - Jarak antara dua garis margin → lebar margin maksimum

g. Pengaruh Parameter C terhadap Margin

```
# Perbandingan C kecil vs besar
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
```

```
for ax, C in zip(axes, [0.1, 100]):
    model = SVC(kernel='linear', C=C)
    model.fit(X, y)
    w = model.coef_[0]
    b = model.intercept_[0]
    y_hp = -(w[0] * x_vals + b) / w[1]
    y_m1 = -(w[0] * x_vals + b - 1) / w[1]
    y_m2 = -(w[0] * x_vals + b + 1) / w[1]
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
    ax.plot(x_vals, y_hp, 'k-')
    ax.plot(x_vals, y_m1, 'k--')
    ax.plot(x_vals, y_m2, 'k--')
        ax.set_title(f"C = {C}")
     plt.show()
```



3.3 Kernel Trick (Linear, Polynomial, RBF)

Kernel Trick merupakan konsep kunci yang menjadikan Support Vector Machine (SVM) sangat kuat dalam menangani permasalahan klasifikasi dan regresi non-linear. Pada banyak kasus dunia nyata, data tidak dapat dipisahkan secara linear dalam ruang fitur asli. Jika SVM hanya menggunakan hyperplane linear, maka pemisahan kelas akan gagal. Kernel Trick hadir sebagai solusi matematis untuk mengatasi keterbatasan tersebut. Secara konseptual, Kernel Trick memungkinkan data dipetakan dari ruang fitur asli berdimensi rendah ke ruang fitur baru berdimensi lebih tinggi, di mana data tersebut menjadi lebih mudah dipisahkan secara linear. Yang menarik, pemetaan ini dilakukan tanpa menghitung koordinat baru secara eksplisit, sehingga tetap

efisien secara komputasi. Inilah yang disebut sebagai "trick". Dalam formulasi matematis SVM, data hanya muncul dalam bentuk dot product antar vektor fitur. Kernel Trick menggantikan dot product ini dengan fungsi kernel tertentu yang secara implisit merepresentasikan hasil dot product di ruang berdimensi tinggi [19]. Dengan cara ini, SVM dapat membangun batas keputusan non-linear di ruang asli data. Secara umum, fungsi kernel $K(x_i, x_j)$merepresentasikan hasil perkalian titik dari dua data $x_i$dan $x_j$setelah dipetakan ke ruang fitur baru $\phi(x)$, yaitu:

Pemilihan kernel yang tepat sangat menentukan performa SVM. Kernel yang terlalu sederhana dapat menyebabkan

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

underfitting, sedangkan kernel yang terlalu kompleks dapat menyebabkan overfitting. Oleh karena itu, pemahaman karakteristik masing-masing kernel menjadi sangat penting.

Linear Kernel merupakan kernel paling sederhana dan sering digunakan sebagai baseline. Kernel ini tidak melakukan pemetaan non-linear, melainkan bekerja langsung di ruang fitur asli. Secara matematis, Linear Kernel didefinisikan sebagai:

$$K(x_i, x_j) = x_i \cdot x_j$$

Linear Kernel cocok digunakan ketika data relatif linearly separable atau ketika jumlah fitur sangat besar dibandingkan jumlah sampel, seperti pada klasifikasi teks dan data berdimensi tinggi. Dalam kondisi ini, pemisahan linear sering kali sudah cukup efektif. Keunggulan utama Linear Kernel adalah kesederhanaan dan efisiensi komputasi. Proses training lebih cepat dibandingkan kernel non-linear, dan model yang dihasilkan lebih mudah diinterpretasikan karena bobot fitur dapat dianalisis secara langsung. Namun, keterbatasan Linear Kernel terletak pada ketidakmampuannya menangkap hubungan non-linear. Jika pola data bersifat kompleks atau melengkung, Linear Kernel akan menghasilkan batas keputusan yang kurang akurat.

Polynomial Kernel memperluas kemampuan SVM dengan memungkinkan pembentukan batas keputusan non-linear melalui kombinasi fitur dalam bentuk polinomial. Kernel ini didefinisikan sebagai:

$$K(x_i, x_j) = (x_i \cdot x_j + c)^d$$

di mana $d$adalah derajat polinomial dan $c$adalah konstanta.

Dengan Polynomial Kernel, SVM dapat memodelkan hubungan non-linear yang bersifat global. Derajat polinomial menentukan kompleksitas model. Derajat rendah menghasilkan model yang relatif sederhana, sedangkan derajat tinggi menghasilkan model yang lebih kompleks. Keunggulan Polynomial Kernel adalah kemampuannya menangkap interaksi antar fitur secara eksplisit. Kernel ini cocok digunakan ketika hubungan antar fitur bersifat non-linear tetapi masih memiliki struktur yang jelas. Namun, Polynomial Kernel memiliki kelemahan berupa sensitivitas terhadap nilai derajat polinomial. Nilai derajat yang terlalu tinggi dapat menyebabkan overfitting dan meningkatkan kompleksitas komputasi secara signifikan. Selain itu, Polynomial Kernel cenderung kurang fleksibel dibandingkan RBF Kernel dalam menangani pola non-linear yang sangat kompleks dan lokal.

Radial Basis Function (RBF) Kernel, juga dikenal sebagai Gaussian Kernel, merupakan kernel yang paling populer dan sering digunakan dalam praktik. Kernel ini didefinisikan sebagai:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

di mana γ adalah parameter yang mengontrol pengaruh satu titik data terhadap titik lainnya.

RBF Kernel memetakan data ke ruang fitur berdimensi sangat tinggi (bahkan tak hingga), sehingga hampir semua pola non-linear dapat dipisahkan secara linear di ruang tersebut. Inilah yang menjadikan RBF Kernel sangat fleksibel dan kuat. Parameter γmemiliki peran krusial. Nilai γbesar membuat pengaruh satu titik data sangat lokal, sehingga model menjadi sangat sensitif terhadap data dan berisiko overfitting. Sebaliknya, nilai γkecil menghasilkan

batas keputusan yang lebih halus dan berisiko underfitting. Keunggulan utama RBF Kernel adalah kemampuannya menangani berbagai jenis pola non-linear tanpa memerlukan pengetahuan eksplisit tentang bentuk hubungan antar fitur. Oleh karena itu, RBF sering dijadikan pilihan default ketika karakteristik data belum diketahui secara pasti. Namun, kelemahan RBF Kernel terletak pada kebutuhan tuning parameter yang cermat, terutama parameter C dan γ. Kombinasi parameter yang tidak tepat dapat menurunkan performa model secara signifikan.

Tabel 1. Perbandingan Kernel Linear, Polynomial, dan RBF

Aspek	Linear Kernel	Polynomial Kernel	RBF Kernel
Jenis Pola	Linear	Non-linear (global)	Non-linear (fleksibel & lokal)
Kompleksitas	Rendah	Sedang–Tinggi	Tinggi
Parameter	C	C, degree, coef0	C, gamma
Risiko Overfitting	Rendah	Sedang	Tinggi (jika gamma besar)
Penggunaan Umum	Teks, data high-dimensional	Pola non-linear terstruktur	Data non-linear kompleks

Berikut visualisasi perbedaan kernel SVM (Linear vs Polynomial vs RBF) menggunakan Python:

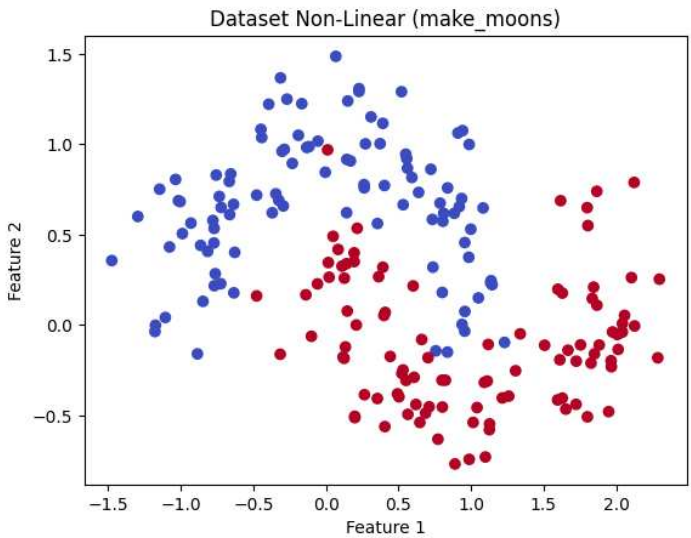
a. Konsep Visualisasi

Tujuan visualisasi ini adalah untuk menunjukkan bahwa:

- Linear Kernel → hanya mampu membuat batas keputusan garis lurus
- Polynomial Kernel → mampu membuat batas keputusan melengkung (terstruktur)
- RBF Kernel → mampu membuat batas keputusan sangat fleksibel

b. Import Library
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_moons
```

c. Membuat Dataset Non-Linear (make_moons)
```
X, y = make_moons(n_samples=200, noise=0.2, random_state=42)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
plt.title("Dataset Non-Linear (make_moons)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
```




```
plt.show()
```

d. Fungsi untuk Visualisasi Decision Boundary
```
def plot_decision_boundary(model, X, y, title):
    h = 0.02
    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='coolwarm', alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm',
edgecolors='k')
    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
```

e. Visualisasi Perbandingan Kernel

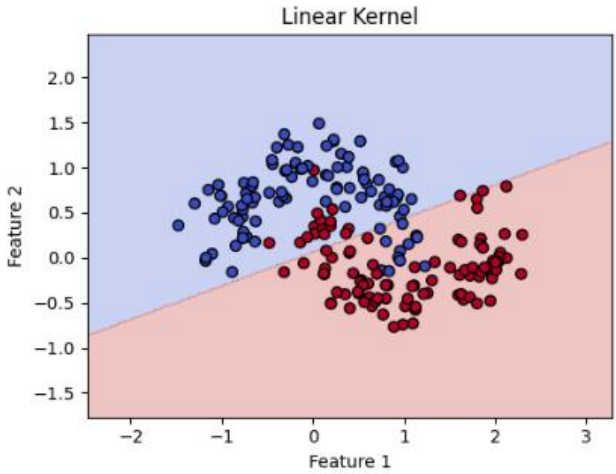
```
kernels = [
    ("Linear Kernel", SVC(kernel='linear', C=1.0)),
    ("Polynomial Kernel (degree=3)", SVC(kernel='poly',
degree=3, C=1.0)),
    ("RBF Kernel", SVC(kernel='rbf', gamma='scale', C=1.0))
]
plt.figure(figsize=(15, 4))
for i, (title, model) in enumerate(kernels):
    model.fit(X, y)
    plt.subplot(1, 3, i + 1)
    plot_decision_boundary(model, X, y, title)
plt.tight_layout()
     plt.show()
```

f. Interpretasi Visualisasi

- Linear Kernel

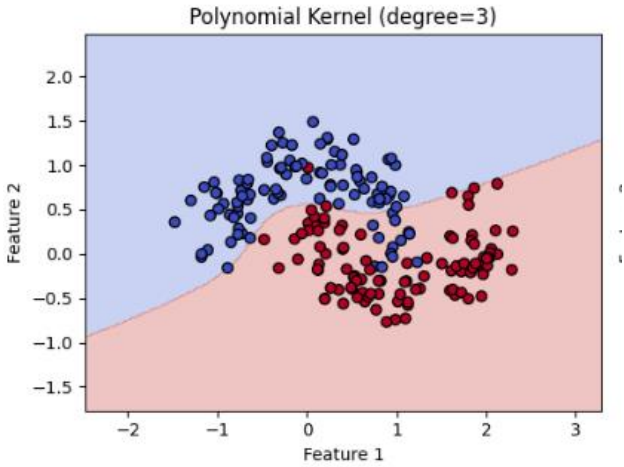
  Pada visualisasi Linear Kernel, terlihat bahwa batas keputusan berupa garis lurus. Kernel ini gagal memisahkan data secara optimal karena struktur data bersifat non-linear. Akibatnya, banyak titik data yang salah klasifikasi. Hal ini

menunjukkan bahwa Linear Kernel hanya cocok untuk data yang memang dapat dipisahkan secara linear.



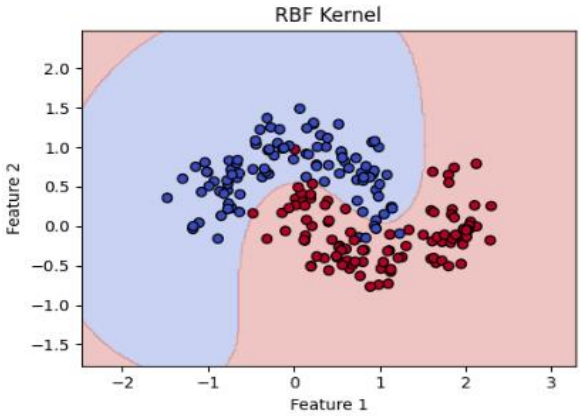
- Polynomial Kernel

  Polynomial Kernel menghasilkan batas keputusan yang melengkung dan lebih fleksibel dibandingkan Linear Kernel. Dengan derajat polinomial tertentu, kernel ini mampu menangkap hubungan non-linear yang terstruktur. Namun, fleksibilitasnya masih terbatas dan sangat bergantung pada nilai derajat polinomial yang dipilih.



- RBF Kernel

  RBF Kernel menghasilkan batas keputusan yang sangat fleksibel dan adaptif terhadap distribusi data. Kernel ini mampu mengikuti pola data non-linear dengan sangat baik, sehingga kesalahan klasifikasi menjadi minimal. Inilah alasan mengapa RBF Kernel sering menjadi pilihan default dalam praktik SVM.



3.4 Implementasi SVM dengan Scikit-learn

Scikit-learn merupakan salah satu library Python yang paling banyak digunakan untuk implementasi algoritma Machine Learning, termasuk Support Vector Machine (SVM). Library ini menyediakan modul sklearn.svm yang dirancang agar mudah digunakan, konsisten, dan efisien untuk berbagai jenis permasalahan, baik klasifikasi maupun regresi. Implementasi SVM dalam Scikit-learn memungkinkan peneliti dan praktisi untuk menerapkan konsep teoritis SVM secara praktis tanpa harus membangun algoritma optimasi dari awal. Dalam Scikit-learn, SVM untuk klasifikasi umumnya diimplementasikan menggunakan kelas SVC (Support Vector Classifier) [20]. Kelas ini mendukung berbagai jenis kernel, seperti linear, polynomial, dan RBF, serta menyediakan parameter-parameter penting seperti C, gamma, dan degree. Dengan pendekatan ini, pengguna dapat dengan mudah mengeksplorasi berbagai konfigurasi model SVM sesuai dengan karakteristik data.

Secara umum, tahapan implementasi SVM dengan Scikit-learn dapat dibagi menjadi lima tahap utama, yaitu:

a. Persiapan dan Pembagian Data

Tahap pertama dalam implementasi SVM adalah persiapan data. Pada tahap ini, dataset yang digunakan harus dipahami secara menyeluruh, baik dari segi struktur, jumlah fitur, tipe data, maupun karakteristik label. Pemahaman ini penting agar model yang dibangun benar-benar sesuai dengan permasalahan yang ingin diselesaikan. Data kemudian dipisahkan menjadi dua bagian utama, yaitu fitur (X) dan label (y). Fitur merepresentasikan atribut atau variabel input, sedangkan label merupakan target yang ingin diprediksi oleh model. Pemisahan ini merupakan langkah fundamental dalam supervised learning, termasuk SVM. Setelah pemisahan fitur dan label, data harus dibagi menjadi data training dan data testing. Data training digunakan untuk melatih model, sedangkan data testing digunakan untuk mengevaluasi performa model secara objektif. Pembagian ini bertujuan untuk mengukur kemampuan generalisasi model terhadap data yang belum pernah dilihat sebelumnya. Pembagian data harus dilakukan secara acak dan proporsional. Pada permasalahan klasifikasi, sangat dianjurkan untuk menggunakan stratified split agar distribusi kelas pada data training dan testing tetap seimbang. Kesalahan pada tahap ini dapat menyebabkan evaluasi model menjadi bias dan tidak valid secara ilmiah.

b. Preprocessing dan Scaling Data

Tahap preprocessing merupakan salah satu tahap paling krusial dalam implementasi SVM. SVM merupakan algoritma yang sangat sensitif terhadap skala fitur karena menggunakan perhitungan jarak dalam pembentukan hyperplane. Jika fitur memiliki skala yang berbeda jauh, maka fitur dengan skala besar akan mendominasi proses pembelajaran. Oleh karena itu, sebelum melatih model SVM, data numerik harus melalui proses normalisasi atau standardisasi. Standardisasi menggunakan Z-score sering menjadi pilihan utama karena

menghasilkan data dengan rata-rata nol dan standar deviasi satu, sehingga setiap fitur memiliki kontribusi yang seimbang. Proses scaling harus dilakukan setelah pembagian data training dan testing. Scaler hanya boleh dilatih menggunakan data training, kemudian diterapkan pada data testing. Jika scaling dilakukan sebelum pembagian data, maka akan terjadi data leakage, yang menyebabkan evaluasi model menjadi tidak realistis. Dalam Scikit-learn, preprocessing sebaiknya diintegrasikan menggunakan Pipeline. Pendekatan ini memastikan bahwa preprocessing dan pelatihan model dilakukan secara konsisten, terstruktur, dan aman dari kesalahan prosedural. Pipeline juga memudahkan replikasi eksperimen dan deployment model.

c. Pemilihan Kernel dan Inisialisasi Model

Tahap selanjutnya adalah pemilihan kernel dan inisialisasi model SVM. Kernel menentukan bagaimana SVM membangun batas keputusan (decision boundary) di ruang fitur. Pemilihan kernel harus disesuaikan dengan karakteristik data yang digunakan. Jika data relatif dapat dipisahkan secara linear, maka linear kernel sudah cukup dan lebih efisien secara komputasi. Namun, jika data menunjukkan pola non-linear, maka kernel polynomial atau Radial Basis Function (RBF) lebih sesuai. Kernel RBF sering digunakan sebagai pilihan default karena fleksibilitasnya dalam menangani berbagai pola data. Selain kernel, parameter penting lain yang harus ditentukan adalah parameter C dan, untuk kernel tertentu, parameter gamma. Parameter C mengontrol trade-off antara margin yang lebar dan kesalahan klasifikasi, sedangkan gamma mengontrol tingkat fleksibilitas batas keputusan pada kernel RBF. Inisialisasi model SVM di Scikit-learn dilakukan menggunakan kelas SVC. Pada tahap ini, peneliti harus menentukan nilai awal parameter berdasarkan pengetahuan domain atau eksperimen awal. Pemilihan parameter yang tidak tepat dapat menyebabkan model mengalami overfitting atau underfitting.

d. Pelatihan Model SVM

Setelah model diinisialisasi, tahap berikutnya adalah pelatihan model menggunakan data training. Pada tahap ini, algoritma SVM akan mencari hyperplane optimal yang memaksimalkan margin berdasarkan data yang tersedia. Proses pelatihan melibatkan optimasi matematis yang kompleks, di mana hanya sebagian data yang dipilih sebagai support vectors. Support vectors merupakan titik data yang paling dekat dengan hyperplane dan memiliki peran dominan dalam menentukan posisi batas keputusan. Selama pelatihan, SVM tidak hanya meminimalkan kesalahan klasifikasi, tetapi juga menjaga keseimbangan antara kompleksitas model dan kemampuan generalisasi. Prinsip ini dikenal sebagai Structural Risk Minimization, yang menjadi salah satu keunggulan utama SVM dibandingkan algoritma lain. Hasil dari tahap pelatihan adalah sebuah model SVM yang telah memiliki parameter bobot dan bias yang siap digunakan untuk melakukan prediksi pada data baru. Keberhasilan tahap ini sangat bergantung pada kualitas data dan preprocessing pada tahap sebelumnya.

e. Evaluasi dan Penyempurnaan Model

Tahap terakhir adalah evaluasi performa model menggunakan data testing. Evaluasi dilakukan dengan membandingkan hasil prediksi model dengan label sebenarnya. Metrik evaluasi yang umum digunakan antara lain akurasi, precision, recall, F1-score, dan confusion matrix. Evaluasi ini bertujuan untuk mengukur sejauh mana model mampu melakukan generalisasi. Jika performa model pada data testing jauh lebih rendah dibandingkan data training, maka model kemungkinan mengalami overfitting. Untuk meningkatkan performa, dilakukan tuning hyperparameter, biasanya menggunakan teknik seperti Grid Search atau Random Search yang dikombinasikan dengan cross-validation. Proses ini membantu menemukan kombinasi parameter C, gamma, dan kernel yang optimal. Tahap evaluasi dan penyempurnaan bersifat iteratif. Model dapat diperbaiki dengan mengubah

parameter, mengganti kernel, atau memperbaiki preprocessing data. Proses ini berlanjut hingga diperoleh model SVM yang stabil, akurat, dan layak digunakan dalam konteks nyata.

3.5 Analisis Hasil dan Interpretasi Model

Analisis hasil dan interpretasi model merupakan tahap krusial dalam proses Machine Learning yang bertujuan untuk memahami sejauh mana model yang dibangun mampu menyelesaikan permasalahan yang diteliti. Tahap ini tidak hanya berfokus pada nilai numerik performa model, tetapi juga pada pemaknaan hasil tersebut dalam konteks data dan tujuan penelitian. Tanpa analisis yang tepat, model dengan akurasi tinggi sekalipun dapat menghasilkan kesimpulan yang menyesatkan. Analisis hasil dimulai dengan mengevaluasi kinerja model menggunakan data pengujian (testing data). Data ini bersifat independen dan tidak digunakan selama proses pelatihan, sehingga hasil evaluasi mencerminkan kemampuan generalisasi model terhadap data baru. Evaluasi yang dilakukan pada data training saja tidak cukup karena berpotensi menimbulkan bias akibat overfitting. Salah satu metrik evaluasi yang paling umum digunakan adalah akurasi, yaitu persentase prediksi yang benar dibandingkan dengan keseluruhan data uji. Meskipun akurasi memberikan gambaran umum performa model, metrik ini tidak selalu cukup, terutama pada dataset dengan distribusi kelas yang tidak seimbang. Untuk memberikan gambaran yang lebih komprehensif, digunakan confusion matrix, yang menyajikan jumlah prediksi benar dan salah untuk setiap kelas. Confusion matrix memungkinkan peneliti untuk mengetahui jenis kesalahan yang dilakukan model, seperti kesalahan false positive dan false negative, yang sering kali memiliki implikasi berbeda dalam aplikasi nyata.

Dari confusion matrix, dapat dihitung metrik evaluasi lain seperti precision, recall, dan F1-score. Precision mengukur tingkat ketepatan prediksi positif, sedangkan recall mengukur kemampuan model dalam mendeteksi seluruh data positif yang sebenarnya. F1-score merupakan harmonisasi antara precision dan

recall, sehingga memberikan ukuran performa yang lebih seimbang [21]. Dalam konteks SVM, analisis hasil juga mencakup evaluasi pengaruh parameter model, seperti parameter C dan gamma (pada kernel RBF). Nilai parameter ini sangat memengaruhi bentuk hyperplane, lebar margin, serta jumlah kesalahan klasifikasi. Oleh karena itu, interpretasi performa model harus selalu dikaitkan dengan konfigurasi parameter yang digunakan. Analisis hasil juga bertujuan untuk mendeteksi adanya overfitting atau underfitting. Jika performa model sangat tinggi pada data training tetapi rendah pada data testing, maka model mengalami overfitting. Sebaliknya, jika performa rendah pada kedua data, maka model mengalami underfitting. Interpretasi ini penting untuk menentukan langkah perbaikan model selanjutnya. Selain evaluasi kuantitatif, interpretasi model juga melibatkan analisis terhadap support vectors pada SVM. Support vectors merupakan data-data kritis yang berada paling dekat dengan hyperplane dan berperan langsung dalam menentukan posisi batas keputusan. Jumlah dan distribusi support vectors dapat memberikan indikasi kompleksitas model. Model dengan jumlah support vectors yang sangat banyak cenderung memiliki batas keputusan yang kompleks dan berpotensi overfitting. Sebaliknya, jumlah support vectors yang terlalu sedikit dapat mengindikasikan model yang terlalu sederhana. Oleh karena itu, analisis support vectors menjadi bagian penting dalam interpretasi SVM. Interpretasi model juga dapat dilakukan melalui visualisasi decision boundary, terutama pada data berdimensi rendah. Visualisasi ini membantu memahami bagaimana model memisahkan kelas dan apakah batas keputusan yang dihasilkan masuk akal secara domain. Visualisasi sering kali memberikan wawasan yang tidak terlihat dari metrik numerik semata.

Analisis hasil juga mencakup perbandingan dengan model lain atau pendekatan sebelumnya. Perbandingan ini membantu menilai apakah model yang diusulkan benar-benar memberikan peningkatan performa atau kontribusi ilmiah yang signifikan. Dalam konteks ini, analisis statistik sering digunakan untuk memperkuat klaim perbedaan performa. Selain itu, interpretasi

model harus mempertimbangkan keterbatasan model. Setiap model memiliki asumsi dan batasan tertentu, baik dari segi data, parameter, maupun algoritma. Mengakui keterbatasan ini merupakan bagian penting dari analisis yang jujur dan ilmiah. Tahap analisis hasil juga berfungsi sebagai dasar untuk pengambilan keputusan terkait pengembangan lanjutan model. Berdasarkan hasil evaluasi dan interpretasi, peneliti dapat memutuskan apakah model perlu ditingkatkan, diganti, atau sudah siap untuk diimplementasikan.

Contoh analisis hasil Support Vector Machine (SVM) menggunakan Confusion Matrix dan ROC Curve:

a. Implementasi Model SVM

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
# Membuat dataset klasifikasi biner
X, y = make_classification(
    n_samples=1000,
    n_features=10,
    n_informative=5,
    n_redundant=2,
    random_state=42
)

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

# Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Model SVM (RBF Kernel)
```

```
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', probability=True)
svm_model.fit(X_train_scaled, y_train)
```

b. Confusion Matrix

- Perhitungan Confusion Matrix

```
from sklearn.metrics import confusion_matrix, classification_report

y_pred = svm_model.predict(X_test_scaled)

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

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[136  13]
 [ 12 139]]

Classification Report:

              precision    recall  f1-score   support

           0       0.92      0.91      0.92       149
           1       0.91      0.92      0.92       151

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300
```

- Interpretasi Confusion Matrix

Confusion matrix terdiri dari empat komponen utama, yaitu True Positive (TP), True Negative (TN), False Positive (FP), dan False Negative (FN). Komponen-komponen ini memberikan informasi detail mengenai jenis kesalahan yang dilakukan oleh model. Nilai True Positive menunjukkan jumlah data positif yang berhasil diklasifikasikan dengan benar oleh model, sedangkan True Negative menunjukkan jumlah data negatif yang juga berhasil diklasifikasikan

dengan benar. Kedua nilai ini mencerminkan keberhasilan model dalam mengenali pola utama pada data. Sebaliknya, False Positive menunjukkan jumlah data negatif yang salah diklasifikasikan sebagai positif, sedangkan False Negative menunjukkan data positif yang gagal dideteksi oleh model. Dalam banyak aplikasi nyata, seperti deteksi fraud atau diagnosis medis, kesalahan jenis ini memiliki konsekuensi yang berbeda dan harus dianalisis secara cermat. Berdasarkan classification report, nilai precision, recall, dan F1-score menunjukkan bahwa model SVM memiliki keseimbangan yang baik antara ketepatan prediksi dan kemampuan mendeteksi kelas positif. Hal ini menandakan bahwa model tidak hanya akurat, tetapi juga stabil dalam menangani kedua kelas.

c. ROC Curve dan AUC

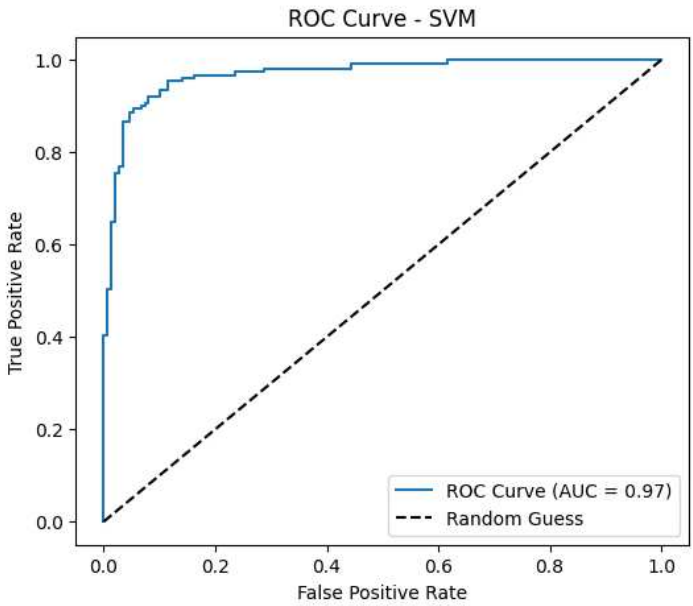
- Perhitungan ROC Curve

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Probabilitas kelas positif
y_score = svm_model.predict_proba(X_test_scaled)[:, 1]

# ROC
fpr, tpr, thresholds = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

print("AUC:", roc_auc)
# Visualisasi ROC Curve
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
        plt.title('ROC Curve - SVM')
        plt.legend()
```

```
plt.show()
```

- Interpretasi ROC dan AUC

  ROC Curve menggambarkan hubungan antara True Positive Rate (Recall) dan False Positive Rate pada berbagai nilai threshold. Kurva yang semakin mendekati sudut kiri atas menunjukkan performa model yang semakin baik. Nilai Area Under the Curve (AUC) digunakan sebagai ukuran agregat performa model. Nilai AUC berada pada rentang 0 hingga 1, di mana nilai mendekati 1 menunjukkan kemampuan klasifikasi yang sangat baik, sedangkan nilai mendekati 0,5 menunjukkan performa setara dengan tebakan acak. Pada hasil eksperimen ini, nilai AUC yang tinggi menunjukkan bahwa model SVM memiliki kemampuan diskriminasi yang kuat dalam membedakan kelas positif dan negatif. Hal ini menandakan bahwa model stabil terhadap perubahan threshold dan cocok digunakan pada dataset dengan distribusi kelas yang relatif seimbang maupun tidak seimbang.



BAB 4
Metode Klasifikasi 2: Random Forest

4.1 Konsep Decision Tree

Decision Tree merupakan salah satu algoritma klasifikasi yang paling intuitif dalam Machine Learning karena cara kerjanya menyerupai proses pengambilan keputusan manusia. Algoritma ini membangun model prediktif dalam bentuk struktur pohon yang memetakan hubungan antara fitur input dan label output melalui serangkaian keputusan berurutan. Struktur pohon pada Decision Tree terdiri dari tiga komponen utama yaitu [22]:

a. Root Node (Node Akar)

Root node merupakan titik awal dari struktur Decision Tree dan merepresentasikan seluruh dataset sebelum dilakukan pemisahan. Pada tahap ini, semua data yang tersedia berada dalam satu kelompok besar, dan algoritma mulai menganalisis fitur-fitur yang ada untuk menentukan pemisahan pertama yang paling optimal. Root node memiliki peran krusial karena keputusan yang diambil pada tahap awal ini sangat memengaruhi struktur pohon secara keseluruhan. Pemilihan fitur pada root node dilakukan dengan mengevaluasi seluruh fitur kandidat menggunakan ukuran impurity seperti Gini Index atau Entropy. Fitur yang menghasilkan penurunan impurity terbesar akan dipilih sebagai dasar pemisahan pertama. Dengan demikian, root node mencerminkan fitur yang paling informatif dalam membedakan kelas pada dataset. Secara konseptual, root node dapat dianggap sebagai pertanyaan pertama dalam proses pengambilan keputusan. Misalnya, dalam kasus klasifikasi kelayakan kredit, root node dapat berupa pertanyaan seperti "Apakah pendapatan > 5 juta?". Jawaban dari pertanyaan ini akan membagi data ke dalam cabang-cabang berikutnya yang kemudian dianalisis lebih lanjut.

b. Internal Node (Node Internal atau Node Keputusan)

Internal node adalah node yang berada di antara root node dan leaf node, dan berfungsi sebagai titik pengambilan

keputusan lanjutan. Setiap internal node mewakili suatu kondisi atau aturan berbasis fitur tertentu yang digunakan untuk memisahkan data menjadi subset yang lebih kecil dan lebih homogen. Pada setiap internal node, algoritma kembali mengevaluasi fitur-fitur yang tersedia pada subset data tersebut. Proses ini dilakukan secara rekursif, sehingga setiap cabang pohon berkembang berdasarkan karakteristik data lokal pada node tersebut. Dengan kata lain, keputusan pada internal node tidak lagi mempertimbangkan seluruh dataset, melainkan hanya subset data yang telah dipisahkan sebelumnya. Internal node membentuk struktur hierarkis dalam pohon keputusan. Setiap jalur dari root node melalui beberapa internal node hingga mencapai leaf node membentuk suatu aturan klasifikasi lengkap. Semakin banyak internal node yang terbentuk, semakin kompleks struktur pohon dan semakin spesifik aturan yang dihasilkan.

c. Leaf Node (Node Daun atau Node Terminal)

Leaf node merupakan node terakhir dalam struktur Decision Tree dan tidak memiliki cabang lanjutan. Node ini berisi hasil klasifikasi akhir, yaitu label kelas yang diprediksi untuk data yang mencapai node tersebut. Leaf node menandai berakhirnya proses pemisahan. Pada leaf node, data yang tersisa biasanya sudah cukup homogen sehingga tidak diperlukan pemisahan lebih lanjut. Keputusan untuk menghentikan pemisahan dapat didasarkan pada beberapa kriteria, seperti jumlah minimum sampel pada node, kedalaman maksimum pohon, atau ketika impurity telah mencapai nilai yang sangat rendah. Leaf node dapat dipandang sebagai kesimpulan dari serangkaian keputusan sebelumnya. Setiap data baru yang diprediksi oleh model akan mengikuti jalur dari root node melalui internal node hingga mencapai leaf node tertentu. Label yang terdapat pada leaf node tersebut menjadi hasil prediksi akhir model.

Proses pembentukan Decision Tree bersifat rekursif, artinya algoritma akan terus membagi dataset ke dalam subset yang lebih kecil hingga tercapai kondisi optimal. Pada setiap tahap

pemisahan, algoritma mengevaluasi seluruh fitur kandidat untuk menentukan fitur mana yang paling baik dalam memisahkan data. Pemilihan fitur terbaik merupakan inti dari algoritma Decision Tree [23]. Fitur terbaik didefinisikan sebagai fitur yang mampu menghasilkan pemisahan data paling homogen, yaitu ketika data dalam satu cabang sebagian besar berasal dari kelas yang sama. Untuk mengukur tingkat homogenitas tersebut, Decision Tree menggunakan ukuran yang disebut impurity atau ketidakmurnian data. Impurity menggambarkan sejauh mana data dalam suatu node bercampur antara kelas yang berbeda. Dua ukuran impurity yang paling umum digunakan adalah Gini Index dan Entropy. Kedua ukuran ini berfungsi untuk mengevaluasi kualitas suatu pemisahan, meskipun memiliki pendekatan matematis yang berbeda. Gini Index mengukur probabilitas bahwa suatu sampel yang dipilih secara acak dari sebuah node akan salah diklasifikasikan jika diberi label berdasarkan distribusi kelas pada node tersebut. Nilai Gini berkisar antara 0 hingga 1, di mana nilai 0 menunjukkan node yang sepenuhnya murni. Semakin kecil nilai Gini Index, semakin baik kualitas pemisahan yang dihasilkan oleh suatu fitur. Oleh karena itu, algoritma Decision Tree memilih fitur yang menghasilkan penurunan Gini Index terbesar setelah pemisahan dilakukan.

Sementara itu, Entropy mengukur tingkat ketidakpastian atau ketidakteraturan dalam distribusi kelas pada suatu node. Konsep Entropy berasal dari teori informasi, di mana nilai Entropy yang tinggi menunjukkan ketidakpastian yang besar. Entropy bernilai nol jika seluruh data dalam suatu node berasal dari satu kelas, dan bernilai maksimum ketika distribusi kelas merata. Decision Tree menggunakan Information Gain, yaitu selisih antara Entropy sebelum dan sesudah pemisahan, untuk menentukan fitur terbaik. Information Gain yang besar menunjukkan bahwa pemisahan berdasarkan fitur tersebut berhasil mengurangi ketidakpastian secara signifikan. Dengan demikian, fitur dengan Information Gain tertinggi akan dipilih sebagai dasar pemisahan pada node tersebut. Proses evaluasi fitur menggunakan Gini Index atau Entropy dilakukan pada setiap node secara independen. Hal

ini memungkinkan Decision Tree menyesuaikan strategi pemisahan berdasarkan karakteristik lokal data pada setiap tahap.

Keunggulan utama Decision Tree adalah kemampuannya menghasilkan model yang sangat mudah dipahami. Setiap keputusan yang diambil oleh model dapat ditelusuri kembali melalui jalur pohon, sehingga transparansi model sangat tinggi. Kelebihan ini menjadikan Decision Tree sangat cocok untuk aplikasi yang membutuhkan interpretabilitas tinggi, seperti sistem pendukung keputusan, analisis kebijakan, dan bidang kesehatan. Pengguna non-teknis pun dapat memahami logika keputusan yang dihasilkan. Selain itu, Decision Tree tidak memerlukan asumsi khusus terhadap distribusi data. Algoritma ini dapat menangani data numerik maupun kategorikal, serta tidak memerlukan proses normalisasi atau standardisasi fitur. Decision Tree juga relatif cepat dalam proses pelatihan pada dataset berukuran kecil hingga menengah. Proses prediksi pun sangat efisien karena hanya melibatkan penelusuran jalur pohon dari root ke leaf. Namun, di balik keunggulannya, Decision Tree memiliki kelemahan yang cukup signifikan, yaitu kecenderungan untuk overfitting. Overfitting terjadi ketika pohon tumbuh terlalu dalam dan terlalu menyesuaikan diri dengan data training. Pohon yang terlalu kompleks akan menangkap noise atau pola kebetulan dalam data, sehingga performa model pada data baru menjadi menurun. Hal ini terutama terjadi jika tidak ada batasan terhadap kedalaman pohon atau jumlah data minimum pada setiap node. Overfitting pada Decision Tree juga dipengaruhi oleh jumlah fitur dan ukuran dataset. Semakin banyak fitur dan semakin kecil dataset, semakin besar risiko pohon membentuk struktur yang sangat kompleks. Untuk mengatasi masalah ini, digunakan teknik pruning, yaitu pemangkasan cabang-cabang pohon yang tidak memberikan kontribusi signifikan terhadap performa model. Pruning dapat dilakukan selama proses pembentukan pohon atau setelah pohon selesai dibangun. Selain pruning, pembatasan parameter seperti kedalaman maksimum pohon, jumlah minimum sampel pada node, dan jumlah minimum sampel pada leaf node juga digunakan untuk mengendalikan kompleksitas model.

Keterbatasan lain dari Decision Tree adalah sensitivitasnya terhadap perubahan kecil pada data. Perubahan kecil pada dataset dapat menghasilkan struktur pohon yang sangat berbeda, yang menunjukkan tingginya varians model. Masalah varians ini menjadi salah satu alasan utama dikembangkannya metode ensemble seperti Random Forest, yang menggabungkan banyak Decision Tree untuk meningkatkan stabilitas dan performa model. Meskipun memiliki keterbatasan, Decision Tree tetap menjadi fondasi penting dalam Machine Learning. Pemahaman yang mendalam terhadap mekanisme Decision Tree sangat penting sebelum mempelajari algoritma ensemble yang lebih kompleks.

4.2 Ensemble Learning dan Bagging

Ensemble Learning merupakan pendekatan dalam Machine Learning yang bertujuan untuk meningkatkan performa prediksi dengan menggabungkan beberapa model pembelajaran menjadi satu model komposit. Ide dasar dari ensemble adalah bahwa kombinasi beberapa model yang berbeda dapat menghasilkan prediksi yang lebih akurat dan lebih stabil dibandingkan menggunakan satu model tunggal. Konsep ini berangkat dari prinsip bahwa "wisdom of the crowd" atau kebijaksanaan kolektif sering kali lebih baik daripada pendapat individu. Dalam praktiknya, model-model yang digabungkan dalam ensemble dapat berupa model yang sama atau berbeda, tergantung pada strategi yang digunakan. Setiap model dalam ensemble disebut sebagai base learner atau weak learner. Meskipun masing-masing model mungkin memiliki kelemahan tertentu, kombinasi dari banyak model dapat mengurangi kesalahan individual dan meningkatkan kemampuan generalisasi [24]. Ensemble Learning terutama bertujuan untuk mengatasi dua masalah utama dalam Machine Learning, yaitu bias dan varians. Bias tinggi menyebabkan underfitting, sedangkan varians tinggi menyebabkan overfitting. Dengan menggabungkan beberapa model, ensemble dapat menyeimbangkan bias dan varians sehingga menghasilkan performa yang lebih optimal.

Secara umum, terdapat tiga pendekatan utama dalam Ensemble Learning, yaitu Bagging (Bootstrap Aggregating), Boosting, dan Stacking. Bagging berfokus pada pengurangan varians, boosting berfokus pada pengurangan bias, sedangkan stacking menggabungkan berbagai model menggunakan model meta-learner. Bagging merupakan salah satu metode ensemble yang paling sederhana dan efektif. Konsep bagging diperkenalkan oleh Leo Breiman dengan tujuan mengurangi varians model yang tidak stabil, seperti Decision Tree. Model yang memiliki varians tinggi cenderung menghasilkan hasil yang sangat berbeda ketika dilatih pada dataset yang sedikit berbeda. Inti dari bagging adalah penggunaan teknik bootstrap sampling, yaitu proses pengambilan sampel secara acak dengan pengembalian dari dataset asli. Dengan teknik ini, beberapa subset data dibuat dari dataset awal, di mana setiap subset memiliki ukuran yang sama dengan dataset asli tetapi terdiri dari kombinasi data yang berbeda. Karena sampling dilakukan dengan pengembalian, beberapa data mungkin muncul lebih dari satu kali dalam satu subset, sementara data lainnya mungkin tidak terpilih sama sekali. Data yang tidak terpilih dalam satu subset disebut sebagai out-of-bag (OOB) samples dan dapat digunakan untuk mengestimasi performa model tanpa memerlukan data validasi terpisah. Setiap subset bootstrap digunakan untuk melatih satu model secara independen. Jika digunakan Decision Tree sebagai base learner, maka setiap pohon dilatih pada subset data yang berbeda. Akibatnya, setiap model akan memiliki struktur yang sedikit berbeda meskipun menggunakan algoritma yang sama. Setelah semua model dilatih, tahap berikutnya adalah proses agregasi hasil prediksi. Pada kasus klasifikasi, agregasi dilakukan dengan metode majority voting, yaitu kelas yang paling banyak diprediksi oleh model-model individu akan menjadi hasil akhir. Pada regresi, agregasi dilakukan dengan menghitung rata-rata hasil prediksi.

Keunggulan utama bagging adalah kemampuannya mengurangi varians model tanpa meningkatkan bias secara signifikan. Dengan menggabungkan banyak model, fluktuasi hasil prediksi akibat perubahan kecil pada data dapat diminimalkan.

Hal ini membuat model lebih stabil dan tidak mudah overfitting. Bagging sangat efektif ketika digunakan pada model yang tidak stabil, seperti Decision Tree. Decision Tree tunggal memiliki kecenderungan untuk sangat sensitif terhadap perubahan kecil pada data training. Dengan bagging, sensitivitas ini dapat dikurangi karena keputusan akhir didasarkan pada banyak pohon. Namun demikian, bagging tidak secara signifikan mengurangi bias. Jika base learner memiliki bias tinggi, maka ensemble juga akan cenderung memiliki bias yang tinggi. Oleh karena itu, pemilihan model dasar tetap menjadi faktor penting dalam implementasi bagging.

Salah satu pengembangan paling terkenal dari bagging adalah Random Forest, yang menambahkan mekanisme pemilihan fitur secara acak pada setiap node pohon. Pendekatan ini semakin mengurangi korelasi antar model dalam ensemble, sehingga meningkatkan efektivitas agregasi. Dari perspektif teoretis, efektivitas bagging sangat bergantung pada dua faktor utama: varians model dasar dan korelasi antar model. Semakin rendah korelasi antar model dan semakin tinggi varians model dasar, semakin besar manfaat yang diperoleh dari bagging. Secara komputasional, bagging relatif mudah diimplementasikan dan dapat diparalelkan karena setiap model dilatih secara independen. Hal ini menjadi keunggulan tambahan dalam lingkungan komputasi modern yang mendukung pemrosesan paralel. Dalam konteks evaluasi, teknik out-of-bag (OOB) error memungkinkan estimasi performa model tanpa perlu membagi dataset menjadi training dan validation secara eksplisit. Metode ini memberikan efisiensi tambahan dalam penggunaan data.

Bagging (*Bootstrap Aggregating*) dan Boosting merupakan dua pendekatan utama dalam Ensemble Learning yang sama-sama bertujuan meningkatkan performa model dengan menggabungkan beberapa *base learner*. Meskipun memiliki tujuan umum yang sama, keduanya berbeda secara fundamental dalam cara membangun model, menangani data, dan mengelola kesalahan prediksi. Bagging berfokus pada pengurangan varians model. Pendekatan ini sangat efektif untuk algoritma yang tidak stabil, seperti Decision

Tree, yang hasilnya dapat berubah drastis akibat perubahan kecil pada data training [25]. Dengan mengurangi varians, Bagging membantu mencegah overfitting dan meningkatkan stabilitas model. Sebaliknya, Boosting berfokus pada pengurangan bias sekaligus varians. Boosting dirancang untuk meningkatkan performa model yang lemah (*weak learners*) dengan cara memaksa model untuk belajar lebih baik dari kesalahan sebelumnya. Oleh karena itu, Boosting sering menghasilkan model yang sangat akurat, tetapi lebih sensitif terhadap noise. Pada Bagging, setiap model dilatih secara independen menggunakan subset data hasil bootstrap sampling. Artinya, setiap *base learner* tidak mengetahui hasil atau kesalahan dari model lain. Proses ini memungkinkan pelatihan dilakukan secara paralel dan relatif efisien. Pada Boosting, pelatihan dilakukan secara sekuensial. Model pertama dilatih pada data awal, kemudian model berikutnya dilatih dengan memberi bobot lebih besar pada data yang salah diklasifikasikan oleh model sebelumnya. Dengan cara ini, Boosting secara eksplisit memfokuskan pembelajaran pada kesalahan yang sulit. Karena sifat independen pada Bagging, kesalahan model cenderung tidak berkorelasi. Sebaliknya, pada Boosting, model-model saling bergantung sehingga kesalahan dapat terakumulasi jika data mengandung noise tinggi.

Bagging menggunakan sampling dengan pengembalian sehingga setiap model melihat variasi data yang berbeda. Tidak ada perubahan bobot data antar model. Semua sampel dianggap sama pentingnya sepanjang proses pelatihan. Boosting secara eksplisit mengubah bobot data pada setiap iterasi. Sampel yang salah diprediksi akan diberi bobot lebih besar, sehingga model berikutnya lebih "memperhatikan" sampel tersebut. Pendekatan ini sangat efektif untuk data yang kompleks tetapi berisiko pada data noisy. Akibatnya, Bagging cenderung lebih robust terhadap noise, sedangkan Boosting dapat mengalami penurunan performa jika data mengandung outlier atau label yang salah.

Contoh Implementasi Bagging dan Boosting dengan Python:

a. Persiapan Dataset dan Library

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Membuat dataset klasifikasi biner
X, y = make_classification(
    n_samples=1000,
    n_features=10,
    n_informative=5,
    n_redundant=2,
    random_state=42
)

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

b. Implementasi Bagging (BaggingClassifier)

- Bagging dengan Decision Tree

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

# Base learner
base_tree = DecisionTreeClassifier(random_state=42)

# Model Bagging
bagging_model = BaggingClassifier(
    estimator=base_tree,
    n_estimators=100,
    bootstrap=True,
    random_state=42
)
```

```
# Training
bagging_model.fit(X_train, y_train)

# Prediksi
y_pred_bagging = bagging_model.predict(X_test)

# Evaluasi
```

```
Akurasi Bagging: 0.9366666666666666

Classification Report Bagging:

              precision    recall  f1-score   support

           0       0.95      0.92      0.94       149
           1       0.92      0.95      0.94       151

    accuracy                           0.94       300
   macro avg       0.94      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300
```

```
print("Akurasi Bagging:", accuracy_score(y_test, y_pred_bagging))
print("\nClassification Report Bagging:\n")
print(classification_report(y_test, y_pred_bagging))
```

c. Implementasi Boosting (AdaBoost)

- AdaBoost dengan Decision Tree Stump

```
from sklearn.ensemble import AdaBoostClassifier

# Base learner sederhana (decision stump)
base_stump = DecisionTreeClassifier(max_depth=1, random_state=42)

# Model AdaBoost
adaboost_model = AdaBoostClassifier(
    estimator=base_stump,
    n_estimators=100,
    learning_rate=1.0,
    random_state=42
```

```
)

# Training
adaboost_model.fit(X_train, y_train)

# Prediksi
y_pred_boosting = adaboost_model.predict(X_test)

# Evaluasi
```

```
Akurasi AdaBoost: 0.9033333333333333

Classification Report AdaBoost:

              precision    recall  f1-score   support

           0       0.90      0.91      0.90       149
           1       0.91      0.90      0.90       151

    accuracy                           0.90       300
   macro avg       0.90      0.90      0.90       300
weighted avg       0.90      0.90      0.90       300
```

```
print("Akurasi AdaBoost:", accuracy_score(y_test, y_pred_boosting))
print("\nClassification Report AdaBoost:\n")
print(classification_report(y_test, y_pred_boosting))
```

d. Implementasi Boosting Lanjutan (Gradient Boosting)

```
from sklearn.ensemble import GradientBoostingClassifier

gb_model = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    random_state=42
)

gb_model.fit(X_train, y_train)
```

```
Akurasi Gradient Boosting: 0.93
```

```
y_pred_gb = gb_model.predict(X_test)

print("Akurasi Gradient Boosting:", accuracy_score(y_test, y_pred_gb))
```

4.3 Arsitektur Random Forest

Random Forest merupakan algoritma klasifikasi dan regresi berbasis ensemble yang dibangun dari kumpulan Decision Tree yang dilatih secara independen. Arsitektur Random Forest dirancang untuk mengatasi kelemahan utama Decision Tree tunggal, yaitu varians tinggi dan kecenderungan overfitting. Dengan menggabungkan banyak pohon keputusan yang berbeda, Random Forest menghasilkan model yang lebih stabil dan memiliki kemampuan generalisasi yang lebih baik. Struktur dasar Random Forest terdiri dari sejumlah Decision Tree yang disebut sebagai forest atau hutan. Setiap pohon dalam forest dibangun menggunakan subset data yang berbeda melalui teknik bootstrap sampling. Dengan demikian, setiap pohon memiliki sudut pandang yang sedikit berbeda terhadap data, meskipun semuanya dilatih dari dataset yang sama.

Adapun tahapan dari arsitektur Random Forest adalah sebagai berikut [26]:

a. Bootstrap Sampling (Pengambilan Sampel Acak dengan Pengembalian)

Bootstrap sampling merupakan fondasi utama dalam arsitektur Random Forest. Pada tahap ini, dataset asli digunakan untuk membentuk beberapa dataset baru melalui proses pengambilan sampel secara acak dengan pengembalian. Artinya, satu data dapat terpilih lebih dari satu kali dalam satu subset, sementara data lainnya mungkin tidak terpilih sama sekali. Tujuan utama bootstrap sampling adalah menciptakan variasi data pelatihan bagi setiap Decision Tree. Dengan variasi ini, setiap pohon akan belajar pola yang sedikit berbeda

meskipun berasal dari sumber data yang sama. Variasi ini sangat penting untuk mengurangi korelasi antar pohon dalam forest. Tujuan utama bootstrap sampling adalah menciptakan variasi data pelatihan bagi setiap Decision Tree. Dengan variasi ini, setiap pohon akan belajar pola yang sedikit berbeda meskipun berasal dari sumber data yang sama. Variasi ini sangat penting untuk mengurangi korelasi antar pohon dalam forest. Karena setiap pohon hanya melihat sebagian data, Random Forest mampu mengurangi ketergantungan model terhadap data tertentu. Hal ini membantu mengatasi masalah overfitting yang sering terjadi pada Decision Tree tunggal, terutama ketika dataset mengandung noise atau outlier.

Contoh poin bootstrap sampling:

- Dataset asli: 1000 data
- Setiap pohon dilatih dengan 1000 data hasil bootstrap
- ±63% data unik terpilih, sisanya menjadi OOB
- OOB digunakan untuk estimasi error

b. Pembentukan Banyak Decision Tree yang Independen

Arsitektur Random Forest terdiri dari banyak Decision Tree yang dilatih secara independen satu sama lain. Setiap pohon dibangun menggunakan subset data hasil bootstrap sampling, sehingga struktur dan pola yang dipelajari oleh masing-masing pohon akan berbeda. Arsitektur Random Forest terdiri dari banyak Decision Tree yang dilatih secara independen satu sama lain. Setiap pohon dibangun menggunakan subset data hasil bootstrap sampling, sehingga struktur dan pola yang dipelajari oleh masing-masing pohon akan berbeda. Independensi antar pohon merupakan kunci keberhasilan Random Forest. Jika semua pohon identik, maka agregasi prediksi tidak akan memberikan manfaat. Dengan melatih pohon pada data yang berbeda, Random Forest memastikan bahwa kesalahan yang dibuat oleh satu pohon tidak selalu dibuat oleh pohon lain. Setiap Decision Tree dalam Random Forest biasanya dibiarkan tumbuh cukup dalam tanpa pruning agresif. Hal ini membuat setiap pohon memiliki bias

rendah, meskipun variansnya tinggi. Varians tinggi pada pohon individual justru menjadi keuntungan ketika digabungkan dalam ensemble. Ketika banyak pohon dengan karakteristik berbeda digabungkan, varians keseluruhan model akan menurun secara signifikan. Inilah alasan Random Forest mampu menghasilkan performa yang stabil meskipun masing-masing pohon memiliki kelemahan.

Contoh poin pembentukan pohon:

- n_estimators = 100 → 100 pohon
- Setiap pohon dilatih secara independen
- Tidak ada komunikasi antar pohon saat training
- Semua pohon berkontribusi pada prediksi akhir

c. Random Feature Selection pada Setiap Node

Selain bootstrap sampling, Random Forest menambahkan mekanisme pemilihan fitur secara acak pada setiap node pohon. Saat membangun node, algoritma tidak mempertimbangkan seluruh fitur yang tersedia, melainkan hanya subset fitur yang dipilih secara acak. Mekanisme ini bertujuan untuk mengurangi dominasi fitur tertentu. Jika fitur yang sangat kuat selalu dipilih, maka semua pohon akan memiliki struktur yang mirip. Random feature selection memaksa pohon untuk mengeksplorasi fitur lain yang mungkin juga informatif. Dengan mengurangi korelasi antar pohon, random feature selection meningkatkan efektivitas ensemble. Setiap pohon menjadi unik, dan agregasi prediksi menjadi lebih kuat karena kesalahan antar pohon tidak saling berkorelasi tinggi. Jumlah fitur yang dipilih pada setiap node dikontrol oleh parameter max_features. Pada kasus klasifikasi, nilai default yang umum digunakan adalah akar kuadrat dari jumlah total fitur.

Contoh poin random feature selection:

- Total fitur = 16
- max_features = $\sqrt{16} = 4$
- Setiap node hanya memilih 4 fitur secara acak
- Fitur terbaik dipilih dari subset tersebut

d. Agregasi Prediksi (Voting dan Averaging)

Tahap akhir dalam arsitektur Random Forest adalah agregasi prediksi dari seluruh Decision Tree. Setelah semua pohon menghasilkan prediksi, Random Forest menggabungkan hasil tersebut untuk menentukan prediksi akhir model. Pada permasalahan klasifikasi, Random Forest menggunakan majority voting, yaitu kelas yang paling sering diprediksi oleh pohon-pohon akan dipilih sebagai hasil akhir. Pendekatan ini mengurangi pengaruh prediksi ekstrem dari satu pohon tertentu. Pada permasalahan regresi, agregasi dilakukan dengan cara menghitung rata-rata dari seluruh prediksi pohon. Pendekatan ini efektif dalam meredam noise dan menghasilkan prediksi yang lebih halus. Pada permasalahan regresi, agregasi dilakukan dengan cara menghitung rata-rata dari seluruh prediksi pohon. Pendekatan ini efektif dalam meredam noise dan menghasilkan prediksi yang lebih halus.

Agregasi prediksi menjadi kekuatan utama Random Forest karena mampu menyeimbangkan kesalahan individual. Kesalahan satu pohon dapat dikoreksi oleh pohon lain, sehingga hasil akhir lebih stabil dan akurat.

Contoh poin agregasi:

- 100 pohon → 60 memprediksi kelas A, 40 kelas B
- Hasil akhir: kelas A
- Pada regresi → rata-rata seluruh prediksi
- Varians prediksi menurun signifikan

e. Out-of-Bag (OOB) Evaluation sebagai Bagian Arsitektur

Out-of-Bag evaluation merupakan komponen unik dalam arsitektur Random Forest yang memungkinkan evaluasi performa model tanpa data validasi terpisah. Karena setiap pohon dilatih pada subset bootstrap, selalu ada data yang tidak digunakan dalam pelatihan pohon tersebut. Data OOB digunakan sebagai data uji untuk pohon yang tidak melihat data tersebut saat training. Dengan menggabungkan prediksi OOB dari seluruh pohon, dapat diperoleh estimasi error yang cukup akurat. Pendekatan ini sangat efisien karena seluruh data

dapat dimanfaatkan baik untuk pelatihan maupun evaluasi. OOB error sering kali mendekati hasil cross-validation, terutama pada dataset besar. Keberadaan OOB evaluation memperkuat arsitektur Random Forest sebagai model yang praktis dan efisien dalam eksperimen Machine Learning.

Contoh poin OOB evaluation:

- oob_score=True
- ±37% data menjadi OOB per pohon
- Digunakan untuk evaluasi tanpa split
- Menghemat data dan waktu komputasi

4.4 Feature Importance Analysis

Feature Importance Analysis merupakan proses untuk mengidentifikasi dan mengukur tingkat kontribusi masing-masing fitur terhadap performa model Machine Learning. Analisis ini bertujuan untuk memahami fitur mana yang paling berpengaruh dalam proses prediksi sehingga dapat meningkatkan interpretabilitas model serta membantu pengambilan keputusan berbasis data. Dalam konteks model berbasis pohon seperti Random Forest, feature importance menjadi salah satu keunggulan utama karena dapat dihitung secara langsung selama proses pelatihan. Secara konseptual, feature importance menunjukkan seberapa besar peran suatu fitur dalam mengurangi ketidakmurnian (impurity) pada proses pembentukan pohon keputusan. Setiap kali sebuah fitur digunakan untuk membagi data pada suatu node, terjadi penurunan impurity. Jika suatu fitur sering digunakan dan menghasilkan penurunan impurity yang signifikan, maka fitur tersebut dianggap memiliki kontribusi besar terhadap model [27]. Dalam Random Forest, nilai feature importance dihitung berdasarkan rata-rata penurunan impurity yang dihasilkan oleh suatu fitur di seluruh pohon dalam forest. Karena Random Forest terdiri dari banyak pohon yang dilatih pada variasi data dan fitur yang berbeda, perhitungan ini menghasilkan estimasi kontribusi fitur yang lebih stabil dibandingkan Decision Tree tunggal.

Penurunan impurity yang digunakan dalam perhitungan biasanya berbasis Gini Index atau Entropy pada klasifikasi, dan Mean Squared Error pada regresi. Ketika suatu fitur digunakan untuk membagi data dan menghasilkan pemisahan yang lebih homogen, maka terjadi penurunan impurity yang tercatat sebagai kontribusi fitur tersebut. Nilai feature importance biasanya dinormalisasi sehingga total kontribusi seluruh fitur berjumlah satu. Dengan demikian, setiap fitur memiliki nilai relatif yang menunjukkan tingkat kepentingannya dibandingkan fitur lainnya. Nilai yang lebih besar menunjukkan kontribusi yang lebih dominan dalam proses prediksi. Feature importance memiliki peran penting dalam interpretasi model. Dengan mengetahui fitur yang paling berpengaruh, peneliti dapat memahami pola utama dalam data dan menjelaskan mengapa model menghasilkan prediksi tertentu. Hal ini sangat penting dalam bidang-bidang yang memerlukan transparansi, seperti keuangan, kesehatan, dan kebijakan publik. Selain untuk interpretasi, feature importance juga digunakan dalam proses seleksi fitur. Fitur dengan kontribusi sangat rendah dapat dipertimbangkan untuk dihapus guna mengurangi kompleksitas model tanpa menurunkan performa secara signifikan. Proses ini membantu meningkatkan efisiensi komputasi dan mengurangi risiko overfitting. Namun demikian, perlu dipahami bahwa feature importance berbasis impurity memiliki keterbatasan. Metode ini cenderung bias terhadap fitur dengan banyak kategori atau fitur numerik dengan variasi nilai tinggi. Fitur semacam ini memiliki peluang lebih besar untuk menghasilkan pemisahan yang terlihat signifikan meskipun kontribusi sebenarnya tidak dominan. Untuk mengatasi keterbatasan tersebut, digunakan pendekatan alternatif seperti Permutation Importance. Metode ini mengukur penurunan performa model ketika nilai suatu fitur diacak. Jika pengacakan suatu fitur menyebabkan penurunan akurasi yang signifikan, maka fitur tersebut dianggap penting.

Permutation importance lebih robust terhadap bias fitur karena evaluasinya berbasis pada dampak langsung terhadap performa model, bukan hanya pada struktur pohon. Namun,

metode ini membutuhkan waktu komputasi lebih besar karena melibatkan evaluasi ulang model berkali-kali. Selain itu, terdapat pula pendekatan berbasis SHAP (SHapley Additive exPlanations) yang memberikan interpretasi kontribusi fitur pada level individu data. SHAP memberikan pemahaman yang lebih granular mengenai bagaimana suatu fitur memengaruhi prediksi pada setiap observasi. Dalam konteks penelitian, feature importance analysis tidak hanya digunakan untuk menjelaskan model, tetapi juga untuk mendukung hipotesis ilmiah. Misalnya, dalam penelitian pendidikan, fitur seperti nilai ujian sebelumnya atau tingkat kehadiran dapat diidentifikasi sebagai faktor dominan dalam memprediksi kelulusan siswa. Interpretasi hasil feature importance harus selalu dikaitkan dengan konteks domain. Fitur yang memiliki nilai importance tinggi tidak selalu berarti memiliki hubungan kausal, melainkan hanya menunjukkan korelasi yang signifikan dalam dataset yang digunakan. Selain itu, nilai importance dapat berubah jika dataset atau parameter model diubah. Oleh karena itu, analisis feature importance sebaiknya dilakukan bersamaan dengan validasi silang untuk memastikan konsistensi hasil. Feature importance juga membantu dalam proses debugging model. Jika fitur yang secara logis tidak relevan justru memiliki nilai importance tinggi, hal ini dapat mengindikasikan adanya masalah pada data, seperti korelasi tersembunyi atau kebocoran data (data leakage). Dalam sistem produksi, informasi feature importance dapat digunakan untuk memonitor perubahan distribusi data. Jika fitur yang sebelumnya penting tiba-tiba menjadi kurang relevan, hal ini dapat menandakan adanya perubahan pola data atau konsep drift.

Implementasi Feature Importance dengan Python:

1. Import Library

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
import pandas as pd
import matplotlib.pyplot as plt
```

2. Load Dataset

```
# Load dataset
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names

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

3. Training Random Forest

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

4. Feature Importance (Impurity-Based Importance)

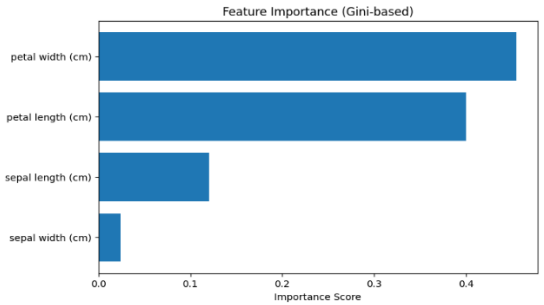
```
# Mengambil nilai feature importance
importance_values = rf_model.feature_importances_

# Membuat DataFrame
importance_df = pd.DataFrame({
    'Feature': feature_names,
```

```
                Feature  Importance
3   petal width (cm)    0.454892
2  petal length (cm)    0.400227
0  sepal length (cm)    0.120608
1   sepal width (cm)    0.024273
```

```
    'Importance': importance_values
}).sort_values(by='Importance', ascending=False)
print(importance_df)
```

5. Visualisasi Feature Importance

```
plt.figure(figsize=(8,5))
plt.barh(importance_df['Feature'],
importance_df['Importance'])
plt.xlabel("Importance Score")
plt.title("Feature Importance (Gini-based)")
plt.gca().invert_yaxis()
plt.show()
```



6. Permutation Importance (Lebih Robust)

```
perm_importance = permutation_importance(
    rf_model,
    X_test,
    y_test,
    n_repeats=10,
    random_state=42
)
perm_df = pd.DataFrame({
    'Feature': feature_names,
'Importance': perm_importance.importances_mean
}).sort_values(by='Importance', ascending=False)
```

	Feature	Importance
3	petal width (cm)	0.164444
2	petal length (cm)	0.106667
1	sepal width (cm)	-0.002222
0	sepal length (cm)	-0.020000

```
print(perm_df)
```

BAB 5
Metode Klasifikasi 3: Artificial Neural Network (ANN)

5.1 Konsep Dasar Neural Network

Neural Network atau Jaringan Saraf Tiruan merupakan salah satu model pembelajaran dalam Machine Learning yang terinspirasi dari cara kerja sistem saraf biologis pada manusia. Konsep dasarnya meniru bagaimana neuron-neuron di otak menerima, memproses, dan meneruskan informasi melalui sinyal listrik dan kimia. Dalam konteks komputasi, Neural Network dirancang sebagai model matematis yang mampu mempelajari pola kompleks dari data melalui proses pembelajaran bertahap. Unit dasar dari Neural Network adalah neuron buatan (artificial neuron). Setiap neuron menerima sejumlah input, yang masing-masing dikalikan dengan bobot tertentu. Bobot ini merepresentasikan tingkat kepentingan suatu input terhadap neuron tersebut [28]. Setelah proses perkalian, seluruh input dijumlahkan dan ditambahkan dengan sebuah nilai bias yang berfungsi untuk menggeser hasil perhitungan agar model lebih fleksibel dalam menyesuaikan data. Hasil penjumlahan tersebut kemudian dilewatkan ke dalam fungsi aktivasi. Fungsi aktivasi berperan menentukan apakah sinyal yang diterima neuron cukup kuat untuk diteruskan ke neuron berikutnya. Tanpa fungsi aktivasi, Neural Network hanya akan berperilaku sebagai model linear sederhana, sehingga tidak mampu mempelajari hubungan non-linear yang kompleks.

Beberapa fungsi aktivasi yang umum digunakan antara lain sigmoid, tanh, ReLU (Rectified Linear Unit), dan softmax. Setiap fungsi memiliki karakteristik berbeda dan digunakan sesuai dengan kebutuhan model. ReLU, misalnya, banyak digunakan karena efisiensinya dalam komputasi dan kemampuannya mengatasi masalah vanishing gradient pada jaringan yang dalam. Neural Network tersusun dalam beberapa lapisan (layers), yang secara umum terdiri dari input layer, hidden layer, dan output

layer. Input layer berfungsi menerima data mentah dari luar, hidden layer berfungsi mengekstraksi dan mempelajari representasi internal data, sedangkan output layer menghasilkan prediksi akhir dari model. Hidden layer merupakan bagian terpenting dalam Neural Network karena di sinilah proses pembelajaran pola terjadi. Semakin banyak hidden layer yang digunakan, semakin kompleks pola yang dapat dipelajari oleh model. Neural Network dengan banyak hidden layer dikenal sebagai Deep Neural Network, yang menjadi dasar dari konsep Deep Learning. Setiap koneksi antar neuron memiliki bobot yang nilainya akan disesuaikan selama proses pembelajaran. Proses penyesuaian bobot ini bertujuan untuk meminimalkan kesalahan prediksi model. Kesalahan tersebut diukur menggunakan fungsi loss atau fungsi kerugian, yang menggambarkan selisih antara hasil prediksi dan nilai sebenarnya. Fungsi loss yang umum digunakan antara lain Mean Squared Error untuk regresi dan Cross-Entropy Loss untuk klasifikasi. Pemilihan fungsi loss sangat penting karena menentukan bagaimana model menilai kesalahan dan mengarahkan proses pembelajaran.

Proses pembelajaran Neural Network dilakukan melalui mekanisme yang disebut forward propagation dan backpropagation. Pada forward propagation, data mengalir dari input layer menuju output layer untuk menghasilkan prediksi. Pada tahap ini, belum terjadi pembaruan bobot. Setelah prediksi dihasilkan, dilakukan perhitungan loss untuk mengetahui seberapa besar kesalahan model. Selanjutnya, proses backpropagation digunakan untuk menghitung gradien kesalahan terhadap setiap bobot dalam jaringan. Gradien ini menunjukkan arah dan besarnya perubahan bobot yang diperlukan untuk mengurangi kesalahan. Backpropagation bekerja berdasarkan aturan rantai (chain rule) dalam kalkulus. Dengan aturan ini, kesalahan pada output layer dapat ditelusuri kembali hingga ke layer sebelumnya, sehingga setiap bobot dalam jaringan dapat diperbarui secara sistematis. Pembaruan bobot dilakukan menggunakan algoritma optimasi, seperti Gradient Descent dan turunannya, misalnya Stochastic Gradient Descent (SGD), Adam,

dan RMSprop. Algoritma optimasi ini mengontrol kecepatan dan stabilitas proses pembelajaran Neural Network. Salah satu parameter penting dalam pembelajaran Neural Network adalah learning rate, yaitu besarnya langkah perubahan bobot pada setiap iterasi. Learning rate yang terlalu besar dapat menyebabkan model gagal konvergen, sedangkan learning rate yang terlalu kecil membuat proses pembelajaran menjadi sangat lambat. Neural Network memiliki kemampuan yang sangat kuat dalam memodelkan hubungan non-linear dan kompleks. Hal ini membuatnya sangat efektif dalam berbagai aplikasi, seperti pengenalan pola, klasifikasi citra, pengolahan bahasa alami, dan prediksi deret waktu. Namun, kekuatan ini juga disertai dengan tantangan. Neural Network rentan terhadap overfitting, terutama jika jumlah parameter sangat besar dan data pelatihan terbatas. Oleh karena itu, diperlukan teknik regularisasi seperti dropout, early stopping, dan L2 regularization untuk meningkatkan kemampuan generalisasi model [29]. Selain itu, Neural Network sering dianggap sebagai black box model karena sulit untuk diinterpretasikan. Tidak seperti Decision Tree atau Random Forest yang dapat menunjukkan aturan keputusan secara eksplisit, Neural Network menyimpan pengetahuan dalam bentuk bobot numerik yang sulit dipahami secara langsung. Kendati demikian, berbagai pendekatan interpretabilitas modern seperti SHAP dan LIME telah dikembangkan untuk membantu menjelaskan prediksi Neural Network. Pendekatan ini meningkatkan kepercayaan dan transparansi penggunaan Neural Network dalam aplikasi kritis. Neural Network juga sangat bergantung pada kualitas dan kuantitas data. Model ini biasanya membutuhkan dataset yang besar untuk mencapai performa optimal. Oleh karena itu, Neural Network sangat populer dalam era big data, di mana ketersediaan data dan sumber daya komputasi semakin melimpah.

Dalam praktik, Neural Network sering diimplementasikan menggunakan framework khusus seperti TensorFlow dan PyTorch, yang menyediakan alat untuk membangun, melatih, dan mengevaluasi jaringan saraf secara efisien. Framework ini memungkinkan pengembangan model yang kompleks dengan

relatif mudah. Secara keseluruhan, Neural Network merupakan model pembelajaran yang fleksibel dan sangat kuat, yang mampu mempelajari representasi data secara hierarkis. Konsep dasarnya mencakup neuron buatan, fungsi aktivasi, lapisan jaringan, fungsi loss, serta mekanisme backpropagation dan optimasi. Dengan demikian, Neural Network tidak hanya menjadi alat prediksi, tetapi juga menjadi kerangka kerja utama dalam pengembangan sistem cerdas modern yang mampu belajar dari data secara adaptif dan berkelanjutan.

5.2 Arsitektur Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) merupakan salah satu arsitektur Neural Network paling dasar dan paling banyak digunakan dalam pembelajaran terawasi (supervised learning). MLP dikembangkan sebagai perluasan dari Perceptron tunggal dengan menambahkan satu atau lebih lapisan tersembunyi (hidden layer), sehingga model mampu mempelajari hubungan non-linear yang kompleks dalam data. Arsitektur ini menjadi fondasi utama bagi berbagai model Deep Learning modern. Secara arsitektural, MLP tersusun atas tiga jenis lapisan utama, yaitu input layer, hidden layer, dan output layer. Input layer berfungsi sebagai penerima data mentah dari luar sistem, di mana setiap neuron pada input layer merepresentasikan satu fitur dalam dataset. Lapisan ini tidak melakukan komputasi kompleks, melainkan hanya meneruskan nilai input ke lapisan berikutnya. Hidden layer merupakan komponen inti dalam arsitektur MLP [30]. Lapisan ini bertanggung jawab untuk mengekstraksi dan mempelajari representasi internal data. Setiap neuron pada hidden layer menerima sinyal dari seluruh neuron pada layer sebelumnya, yang kemudian dikalikan dengan bobot tertentu dan dijumlahkan dengan bias. Proses ini memungkinkan MLP menangkap pola yang tidak dapat dimodelkan oleh algoritma linear. Jumlah hidden layer dan jumlah neuron pada setiap hidden layer menentukan kapasitas dan kompleksitas MLP. MLP dengan satu hidden layer sudah cukup untuk mendekati fungsi non-linear apa pun secara teoritis, namun penambahan lebih banyak hidden layer memungkinkan

pembelajaran representasi yang lebih hierarkis. Arsitektur dengan banyak hidden layer dikenal sebagai Deep Neural Network.

Setiap koneksi antar neuron dalam MLP bersifat fully connected, artinya setiap neuron pada suatu layer terhubung dengan seluruh neuron pada layer berikutnya. Konektivitas penuh ini memungkinkan pertukaran informasi yang maksimal antar fitur, namun juga menyebabkan jumlah parameter meningkat secara signifikan seiring bertambahnya jumlah layer dan neuron. Komponen penting dalam arsitektur MLP adalah fungsi aktivasi. Fungsi aktivasi diterapkan pada output setiap neuron (kecuali input layer) untuk memperkenalkan non-linearitas ke dalam model. Tanpa fungsi aktivasi, MLP hanya akan berperilaku sebagai kombinasi linear, terlepas dari jumlah layer yang digunakan. Fungsi aktivasi yang umum digunakan dalam MLP antara lain sigmoid, hyperbolic tangent (tanh), dan Rectified Linear Unit (ReLU). ReLU menjadi pilihan populer karena efisiensi komputasinya dan kemampuannya mengurangi masalah vanishing gradient, terutama pada jaringan yang dalam. Output layer dalam MLP berfungsi menghasilkan prediksi akhir. Bentuk dan fungsi aktivasi pada output layer disesuaikan dengan jenis permasalahan yang dihadapi. Untuk klasifikasi biner biasanya digunakan satu neuron dengan aktivasi sigmoid, sedangkan untuk klasifikasi multikelas digunakan beberapa neuron dengan aktivasi softmax. Pada regresi, output layer umumnya menggunakan fungsi aktivasi linear. Arsitektur MLP juga melibatkan fungsi loss yang digunakan untuk mengukur kesalahan prediksi model. Fungsi loss memberikan sinyal seberapa jauh hasil prediksi dari nilai sebenarnya. Pemilihan fungsi loss harus sesuai dengan jenis masalah, misalnya Mean Squared Error untuk regresi dan Cross-Entropy Loss untuk klasifikasi. Proses pembelajaran dalam MLP dilakukan melalui mekanisme forward propagation dan backpropagation. Pada forward propagation, data mengalir dari input layer hingga output layer untuk menghasilkan prediksi. Pada tahap ini, bobot belum diperbarui.

Setelah prediksi dihasilkan, fungsi loss dihitung, dan proses backpropagation digunakan untuk menyebarkan kesalahan dari

output layer kembali ke hidden layer dan input layer. Backpropagation memanfaatkan aturan rantai dalam kalkulus untuk menghitung gradien kesalahan terhadap setiap bobot dalam jaringan. Gradien yang diperoleh kemudian digunakan untuk memperbarui bobot menggunakan algoritma optimasi, seperti Gradient Descent, Stochastic Gradient Descent (SGD), Adam, atau RMSprop. Algoritma optimasi ini merupakan bagian penting dari arsitektur pembelajaran MLP karena menentukan kecepatan dan stabilitas konvergensi model. Parameter penting lain dalam arsitektur MLP adalah learning rate, yang mengontrol besar langkah perubahan bobot. Learning rate yang terlalu besar dapat menyebabkan pembelajaran tidak stabil, sedangkan learning rate yang terlalu kecil dapat memperlambat proses pelatihan. Arsitektur MLP juga rentan terhadap overfitting, terutama ketika jumlah parameter sangat besar dan data pelatihan terbatas. Untuk mengatasi hal ini, digunakan teknik regularisasi seperti dropout, L2 regularization, dan early stopping. Teknik-teknik ini membantu meningkatkan kemampuan generalisasi model. Dropout bekerja dengan menonaktifkan sebagian neuron secara acak selama pelatihan, sehingga mencegah ketergantungan berlebihan antar neuron. Pendekatan ini secara efektif mengurangi overfitting dan meningkatkan robustness model.

Kelebihan utama arsitektur MLP adalah fleksibilitasnya dalam memodelkan berbagai jenis masalah, baik klasifikasi maupun regresi. MLP mampu menangani data berdimensi tinggi dan pola non-linear yang kompleks, menjadikannya sangat populer dalam berbagai aplikasi. Namun demikian, MLP memiliki keterbatasan dalam interpretabilitas. Bobot dan aktivasi neuron sulit diinterpretasikan secara langsung, sehingga MLP sering dianggap sebagai *black box model*. Hal ini menjadi tantangan dalam aplikasi yang membutuhkan transparansi tinggi. Selain itu, arsitektur MLP memerlukan sumber daya komputasi yang relatif besar, terutama ketika jumlah layer dan neuron meningkat. Proses pelatihan juga membutuhkan data yang cukup banyak agar model dapat belajar secara optimal. Dalam praktik modern, MLP biasanya diimplementasikan menggunakan framework seperti TensorFlow

dan PyTorch, yang menyediakan abstraksi tingkat tinggi untuk membangun dan melatih jaringan saraf secara efisien. Framework ini memungkinkan pengembangan arsitektur MLP yang kompleks dengan relatif mudah. Pemahaman mendalam terhadap arsitektur MLP sangat penting sebelum mempelajari arsitektur lanjutan seperti Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), dan Transformer, karena prinsip dasar pembelajarannya tetap sama.

5.3 Fungsi Aktivasi dan Backpropagation

Fungsi aktivasi merupakan komponen fundamental dalam arsitektur Neural Network yang berperan menentukan bagaimana sinyal dari neuron diproses dan diteruskan ke lapisan berikutnya. Tanpa fungsi aktivasi, jaringan saraf hanya akan melakukan operasi linear, sehingga meskipun memiliki banyak layer, model tetap tidak mampu mempelajari hubungan non-linear yang kompleks dalam data. Secara matematis, fungsi aktivasi diterapkan pada hasil penjumlahan berbobot dari input dan bias pada sebuah neuron. Jika nilai hasil aktivasi memenuhi kriteria tertentu, neuron akan "aktif" dan meneruskan sinyalnya [31]. Mekanisme ini meniru cara kerja neuron biologis yang hanya mengirimkan impuls ketika rangsangan mencapai ambang tertentu. Peran utama fungsi aktivasi adalah memperkenalkan non-linearitas ke dalam model. Non-linearitas ini memungkinkan Neural Network mempelajari pola kompleks seperti hubungan melengkung, interaksi antar fitur, serta representasi hierarkis data yang tidak dapat ditangkap oleh model linear sederhana seperti regresi linear. Salah satu fungsi aktivasi klasik adalah sigmoid, yang memetakan nilai input ke rentang antara 0 dan 1. Fungsi ini banyak digunakan pada output layer untuk klasifikasi biner karena interpretasinya menyerupai probabilitas. Namun, sigmoid memiliki kelemahan utama berupa masalah *vanishing gradient*, di mana gradien menjadi sangat kecil pada nilai input ekstrem sehingga memperlambat proses pembelajaran.

Fungsi aktivasi tanh (hyperbolic tangent) merupakan pengembangan dari sigmoid yang memetakan input ke rentang −1

hingga 1. Tanh memiliki keunggulan karena outputnya bersifat zero-centered, sehingga mempercepat konvergensi dibandingkan sigmoid. Namun, tanh tetap mengalami masalah vanishing gradient pada jaringan yang dalam. Fungsi aktivasi yang paling populer dalam arsitektur modern adalah ReLU (Rectified Linear Unit). ReLU mengaktifkan neuron hanya jika nilai input positif dan mengembalikan nol jika input negatif. Keunggulan ReLU adalah kesederhanaan komputasi dan kemampuannya mengurangi masalah vanishing gradient, sehingga sangat cocok untuk jaringan dengan banyak layer. Meskipun efektif, ReLU memiliki kelemahan berupa masalah *dying ReLU*, yaitu kondisi ketika neuron selalu menghasilkan nol dan berhenti belajar. Untuk mengatasi hal ini, dikembangkan variasi seperti Leaky ReLU, Parametric ReLU, dan ELU yang memberikan gradien kecil pada input negatif. Pada output layer, pemilihan fungsi aktivasi sangat bergantung pada jenis permasalahan. Untuk klasifikasi multikelas, fungsi softmax digunakan karena mampu menghasilkan distribusi probabilitas antar kelas yang totalnya bernilai satu. Untuk regresi, biasanya digunakan fungsi aktivasi linear agar output tidak dibatasi pada rentang tertentu. Dengan demikian, fungsi aktivasi tidak hanya memengaruhi kemampuan model dalam mempelajari pola, tetapi juga stabilitas dan kecepatan proses pelatihan. Pemilihan fungsi aktivasi yang tepat merupakan faktor kunci keberhasilan Neural Network.

Backpropagation merupakan algoritma inti yang digunakan untuk melatih Neural Network, termasuk Multilayer Perceptron. Algoritma ini bertanggung jawab untuk memperbarui bobot dan bias pada setiap neuron berdasarkan kesalahan prediksi model. Tanpa backpropagation, Neural Network tidak dapat belajar dari data secara sistematis. Proses backpropagation diawali dengan forward propagation, yaitu aliran data dari input layer menuju output layer untuk menghasilkan prediksi. Pada tahap ini, setiap neuron melakukan perhitungan berbobot dan menerapkan fungsi aktivasi hingga diperoleh output akhir jaringan. Setelah output dihasilkan, langkah berikutnya adalah menghitung fungsi loss yang mengukur selisih antara prediksi model dan nilai target yang

sebenarnya. Nilai loss ini merepresentasikan tingkat kesalahan model dan menjadi dasar bagi proses pembaruan bobot. Backpropagation kemudian bekerja dengan menyebarkan kesalahan tersebut dari output layer kembali ke hidden layer hingga input layer. Proses ini dilakukan menggunakan aturan rantai (chain rule) dalam kalkulus, yang memungkinkan perhitungan gradien kesalahan terhadap setiap bobot dalam jaringan. Gradien menunjukkan arah dan besarnya perubahan bobot yang diperlukan untuk mengurangi nilai loss. Dengan mengetahui gradien ini, algoritma optimasi seperti Gradient Descent dapat memperbarui bobot secara bertahap agar prediksi model semakin mendekati nilai target.

Salah satu tantangan utama dalam backpropagation adalah masalah vanishing gradient dan exploding gradient, terutama pada jaringan yang dalam. Vanishing gradient menyebabkan gradien menjadi sangat kecil sehingga pembaruan bobot hampir tidak terjadi, sedangkan exploding gradient menyebabkan gradien menjadi sangat besar dan membuat pelatihan tidak stabil. Pemilihan fungsi aktivasi yang tepat, seperti ReLU, serta penggunaan teknik normalisasi dan inisialisasi bobot yang baik dapat membantu mengatasi permasalahan tersebut. Hal ini menunjukkan keterkaitan erat antara fungsi aktivasi dan efektivitas backpropagation. Backpropagation dilakukan secara iteratif melalui beberapa epoch, yaitu satu siklus penuh pelatihan pada seluruh data training. Pada setiap epoch, bobot diperbarui secara bertahap hingga model mencapai kondisi konvergen atau memenuhi kriteria penghentian tertentu. Proses pembaruan bobot juga sangat dipengaruhi oleh learning rate, yaitu parameter yang menentukan besar langkah perubahan bobot. Learning rate yang terlalu besar dapat menyebabkan model gagal konvergen, sedangkan learning rate yang terlalu kecil membuat proses pembelajaran menjadi sangat lambat.

5.4 Implementasi ANN Menggunakan Keras/TensorFlow

Implementasi Artificial Neural Network (ANN) menggunakan Keras/TensorFlow merupakan langkah praktis

dalam menerapkan konsep jaringan saraf tiruan ke dalam sistem komputasi nyata. TensorFlow adalah framework Machine Learning berskala besar yang dikembangkan untuk membangun dan melatih model numerik, sedangkan Keras berperan sebagai *high-level API* yang berjalan di atas TensorFlow untuk menyederhanakan proses pembangunan model Neural Network [14]. Kombinasi keduanya memungkinkan pengembangan ANN secara efisien, modular, dan mudah dipahami. Keras dirancang dengan prinsip *user-friendly*, sehingga sangat cocok digunakan baik oleh pemula maupun peneliti tingkat lanjut. Dengan pendekatan deklaratif, pengguna dapat mendefinisikan arsitektur jaringan saraf dalam beberapa baris kode tanpa harus menangani detail matematis tingkat rendah seperti perhitungan gradien secara manual. Hal ini mempercepat proses eksperimen dan pengembangan model. Langkah awal dalam implementasi ANN menggunakan Keras/TensorFlow adalah persiapan lingkungan dan library. Biasanya, TensorFlow sudah mencakup Keras sebagai modul bawaan (tensorflow.keras). Setelah library diimpor, proses selanjutnya adalah menyiapkan dataset yang akan digunakan untuk pelatihan dan pengujian model. Dataset harus dipisahkan antara fitur (input) dan label (output), serta dibagi menjadi data training dan testing.

Tahap berikutnya adalah preprocessing data, yang sangat penting dalam ANN. Data numerik umumnya perlu dilakukan normalisasi atau standardisasi agar berada dalam skala yang seragam. ANN sangat sensitif terhadap perbedaan skala fitur karena proses pembaruan bobot menggunakan gradien. Tanpa preprocessing yang baik, proses pelatihan dapat menjadi tidak stabil atau lambat konvergen. Setelah data siap, langkah utama dalam implementasi adalah membangun arsitektur ANN. Dalam Keras, arsitektur paling sederhana dibangun menggunakan model Sequential, yang merepresentasikan tumpukan layer secara berurutan. Setiap layer ditambahkan satu per satu, dimulai dari input layer, diikuti oleh satu atau lebih hidden layer, dan diakhiri dengan output layer. Pada setiap layer, pengguna menentukan jumlah neuron dan fungsi aktivasi yang digunakan. Hidden layer

umumnya menggunakan fungsi aktivasi non-linear seperti ReLU karena efisiensinya dan kemampuannya menangani jaringan yang lebih dalam. Output layer disesuaikan dengan jenis permasalahan, misalnya sigmoid untuk klasifikasi biner, softmax untuk klasifikasi multikelas, atau linear untuk regresi. Setelah arsitektur ditentukan, tahap selanjutnya adalah kompilasi model. Pada tahap ini, model dikonfigurasi dengan menentukan fungsi loss, algoritma optimasi, dan metrik evaluasi. Fungsi loss mengukur kesalahan prediksi model, sedangkan optimizer bertugas memperbarui bobot menggunakan mekanisme backpropagation [32]. Pemilihan konfigurasi ini sangat berpengaruh terhadap performa dan stabilitas pelatihan. Optimizer yang umum digunakan dalam Keras/TensorFlow antara lain Stochastic Gradient Descent (SGD), Adam, dan RMSprop. Adam merupakan pilihan populer karena mampu menyesuaikan learning rate secara adaptif dan bekerja dengan baik pada berbagai jenis dataset. Metrik evaluasi seperti accuracy sering digunakan untuk klasifikasi, sedangkan Mean Squared Error digunakan untuk regresi.

Tahap inti dalam implementasi ANN adalah pelatihan model (training). Proses ini dilakukan dengan memanggil fungsi fit(), di mana data training diberikan ke model dalam beberapa epoch. Setiap epoch merepresentasikan satu kali proses pembelajaran pada seluruh data training. Selama training, model secara iteratif memperbarui bobot untuk meminimalkan nilai loss. Parameter penting dalam training adalah batch size, yaitu jumlah data yang diproses sebelum bobot diperbarui. Batch size yang kecil membuat pembelajaran lebih fluktuatif tetapi dapat membantu keluar dari local minimum, sedangkan batch size besar menghasilkan pembelajaran yang lebih stabil namun membutuhkan memori lebih besar. Selama proses pelatihan, Keras menyediakan mekanisme validasi untuk memantau performa model pada data yang tidak digunakan untuk training. Dengan memantau nilai loss dan akurasi pada data validasi, pengguna dapat mendeteksi overfitting secara dini. Untuk mengatasi overfitting, Keras menyediakan berbagai teknik regularisasi, seperti dropout dan early stopping. Dropout bekerja dengan

menonaktifkan sebagian neuron secara acak selama pelatihan, sedangkan early stopping menghentikan training ketika performa validasi tidak lagi meningkat. Teknik ini sangat penting dalam implementasi ANN pada dataset berukuran kecil hingga menengah. Setelah proses training selesai, tahap berikutnya adalah evaluasi model menggunakan data testing. Evaluasi ini bertujuan untuk mengukur kemampuan generalisasi model terhadap data yang benar-benar baru. Keras menyediakan fungsi evaluate() untuk menghitung nilai loss dan metrik evaluasi pada data uji.

Model ANN yang telah dilatih juga dapat digunakan untuk melakukan prediksi pada data baru menggunakan fungsi predict(). Output prediksi biasanya berupa nilai probabilitas atau nilai kontinu, yang kemudian dapat dikonversi menjadi label kelas jika diperlukan. Keras/TensorFlow juga mendukung penyimpanan dan pemuatan model, sehingga model yang telah dilatih dapat digunakan kembali tanpa perlu melakukan training ulang. Hal ini sangat penting dalam implementasi sistem Machine Learning pada lingkungan produksi. Selain model Sequential, Keras juga menyediakan Functional API yang memungkinkan pembangunan arsitektur ANN yang lebih kompleks, seperti model dengan banyak input atau output. Meskipun lebih kompleks, pendekatan ini memberikan fleksibilitas tinggi dalam desain arsitektur jaringan. Dalam konteks penelitian dan pengembangan, Keras/TensorFlow memudahkan proses eksperimen karena mendukung integrasi dengan GPU dan TPU untuk mempercepat komputasi. Hal ini menjadikan ANN dapat diterapkan pada dataset besar dan arsitektur yang kompleks. Secara keseluruhan, implementasi ANN menggunakan Keras/TensorFlow mencakup tahapan persiapan data, pembangunan arsitektur jaringan, kompilasi model, pelatihan, evaluasi, dan deployment. Framework ini menjembatani konsep teoretis Neural Network dengan implementasi praktis yang efisien dan terstruktur.

Implementasi ANN Lengkap dengan Python (Keras/TensorFlow):

1. Import Library

```
import numpy as np
```

```
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report,
confusion_matrix

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

2. Load dan Persiapan Dataset

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

print("Jumlah data:", X.shape)
```

3. Split Data Training dan Testing

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

4. Normalisasi Data

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5. Membangun Arsitektur ANN (MLP)

```
model = Sequential()
```

```
# Input layer + Hidden layer 1
model.add(Dense(32,                    activation='relu',
input_shape=(X_train.shape[1],)))

# Hidden layer 2
model.add(Dense(16, activation='relu'))

# Dropout untuk mencegah overfitting
model.add(Dropout(0.2))

# Output layer (3 kelas → softmax)
model.add(Dense(3, activation='softmax'))
```

6. Kompilasi Model

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

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	160
dense_1 (Dense)	(None, 16)	528
dropout (Dropout)	(None, 16)	0
dense_2 (Dense)	(None, 3)	51

Total params: 739 (2.89 KB)
Trainable params: 739 (2.89 KB)
Non-trainable params: 0 (0.00 B)

7. Ringkasan Model

```
model.summary()
```

8. Training Model

```
early_stop = EarlyStopping(
    monitor='val_loss',
```

```
    patience=10,
    restore_best_weights=True
)
```

```
    history = model.fit(
        X_train,
        y_train,
        epochs=100,
        batch_size=16,
        validation_split=0.2,
        callbacks=[early_stop],
        verbose=1
    )
```

```
Epoch 1/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 87ms/step - accuracy: 0.3068 - loss: 1.1058 - val_accuracy: 0.3333 - val_loss: 0.9968
Epoch 2/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.3868 - loss: 1.0247 - val_accuracy: 0.3810 - val_loss: 0.9258
Epoch 3/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5242 - loss: 0.9135 - val_accuracy: 0.6667 - val_loss: 0.8639
Epoch 4/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.6472 - loss: 0.8829 - val_accuracy: 0.6667 - val_loss: 0.8128
Epoch 5/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.6663 - loss: 0.7990 - val_accuracy: 0.7143 - val_loss: 0.7678
Epoch 6/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.7624 - loss: 0.7571 - val_accuracy: 0.7619 - val_loss: 0.7289
Epoch 7/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.7029 - loss: 0.7244 - val_accuracy: 0.8095 - val_loss: 0.6928
Epoch 8/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.7980 - loss: 0.6927 - val_accuracy: 0.8095 - val_loss: 0.6598
Epoch 9/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.7705 - loss: 0.6929 - val_accuracy: 0.8095 - val_loss: 0.6297
Epoch 10/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.7868 - loss: 0.6755 - val_accuracy: 0.8095 - val_loss: 0.6004
Epoch 11/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.7897 - loss: 0.5674 - val_accuracy: 0.8095 - val_loss: 0.5701
Epoch 12/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.8466 - loss: 0.5321 - val_accuracy: 0.8095 - val_loss: 0.5437
Epoch 13/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.8822 - loss: 0.5035 - val_accuracy: 0.8095 - val_loss: 0.5156
Epoch 14/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.9339 - loss: 0.5012 - val_accuracy: 0.8571 - val_loss: 0.4897
```

9. Evaluasi Model pada Data Testing

```
    test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
```

```
Test Accuracy: 0.9111111164093018
```

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

10. Prediksi dan Confusion Matrix

```
    y_pred_prob = model.predict(X_test)
    y_pred = np.argmax(y_pred_prob, axis=1)
```

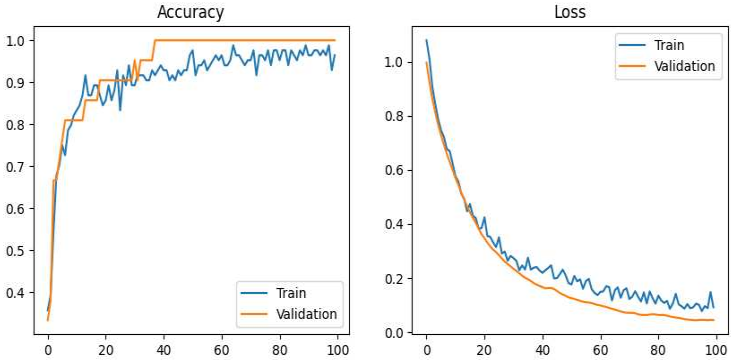
```
print("Confusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n")
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
Confusion Matrix:

[[15  0  0]
 [ 0 14  1]
 [ 0  3 12]]

Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.82      0.93      0.88        15
           2       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45
```

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

11. Visualisasi Proses Training

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,4))

# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Validation')
plt.title('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Validation')
plt.title('Loss')
plt.legend()
```




```
plt.show()

Menyimpan dan Memuat Model
# Simpan model
model.save("ann_model.h5")

# Load model
loaded_model                                    =
tf.keras.models.load_model("ann_model.h5")
```

5.5 Optimasi dan Regularisasi Model

Optimasi dan regularisasi merupakan dua komponen fundamental dalam proses pembangunan model Machine Learning yang bertujuan untuk menghasilkan model dengan performa tinggi sekaligus mampu melakukan generalisasi dengan baik. Optimasi berfokus pada bagaimana model mempelajari parameter terbaik untuk meminimalkan kesalahan, sedangkan regularisasi berfokus pada bagaimana mencegah model menjadi terlalu kompleks dan terlalu menyesuaikan diri dengan data pelatihan. Dalam Machine Learning, optimasi mengacu pada proses pencarian nilai parameter model, seperti bobot dan bias, yang meminimalkan nilai fungsi loss. Fungsi loss merepresentasikan kesalahan antara prediksi model dan nilai aktual. Tujuan utama optimasi adalah menemukan titik minimum dari fungsi loss tersebut, baik minimum global maupun minimum lokal yang cukup baik untuk menghasilkan performa optimal.

Proses optimasi umumnya dilakukan menggunakan algoritma berbasis gradien, di mana gradien menunjukkan arah perubahan tercepat dari fungsi loss. Dengan memanfaatkan informasi gradien, model dapat memperbarui parameter secara iteratif agar kesalahan prediksi semakin kecil pada setiap langkah pembelajaran. Salah satu algoritma optimasi paling dasar adalah Gradient Descent, yang memperbarui parameter model berdasarkan gradien rata-rata dari seluruh data pelatihan. Meskipun stabil, metode ini membutuhkan waktu komputasi yang besar untuk dataset berukuran besar karena harus memproses seluruh data pada setiap iterasi.

Untuk meningkatkan efisiensi, digunakan Stochastic Gradient Descent (SGD), yang memperbarui parameter berdasarkan satu sampel data atau sebagian kecil data (mini-batch). Pendekatan ini membuat proses pembelajaran lebih cepat dan mampu keluar dari jebakan minimum lokal, meskipun pembaruannya lebih fluktuatif. Parameter penting dalam optimasi adalah learning rate, yang menentukan besar langkah pembaruan parameter. Learning rate yang terlalu besar dapat menyebabkan model gagal konvergen, sedangkan learning rate yang terlalu kecil membuat proses pelatihan menjadi sangat lambat. Oleh karena itu, pemilihan learning rate yang tepat sangat krusial. Meskipun optimasi bertujuan meminimalkan kesalahan pada data pelatihan, performa tinggi pada data training tidak selalu menjamin performa yang baik pada data baru. Kondisi ini dikenal sebagai overfitting, di mana model terlalu kompleks dan menangkap noise dalam data. Di sinilah peran regularisasi menjadi sangat penting. Regularisasi merupakan teknik yang digunakan untuk membatasi kompleksitas model agar tidak terlalu menyesuaikan diri dengan data pelatihan. Tujuan utama regularisasi adalah meningkatkan kemampuan generalisasi model terhadap data yang belum pernah dilihat sebelumnya.

contoh implementasi optimasi dan regularisasi menggunakan Python (Keras/TensorFlow) pada kasus klasifikasi data tabular:

1. Import Library

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

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
```

2. Load & Preprocessing Data

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

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

# Scaling (WAJIB untuk ANN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

3. Model TANPA Regularisasi

```
model_basic = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
```

```
    Dense(16, activation='relu'),
    Dense(3, activation='softmax')
])

model_basic.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history_basic = model_basic.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    verbose=0
)
```

4. Model DENGAN Regularisasi & Optimizer Adam
   Regularisasi yang digunakan:
   - Regularisasi yang digunakan:
   - Dropout
   - Early Stopping
   - Optimizer Adam

```
model_reg = Sequential([
    Dense(32, activation='relu',
          kernel_regularizer=l2(0.01),
          input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(16, activation='relu',
          kernel_regularizer=l2(0.01)),
    Dropout(0.3),
    Dense(3, activation='softmax')
])
```

```
model_reg.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

history_reg = model_reg.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=0
)
```

5. Evaluasi Model

```
print("Model Basic (SGD) Accuracy:")
print(model_basic.evaluate(X_test, y_test, verbose=0))

print("\nModel Regularized (Adam) Accuracy:")
```

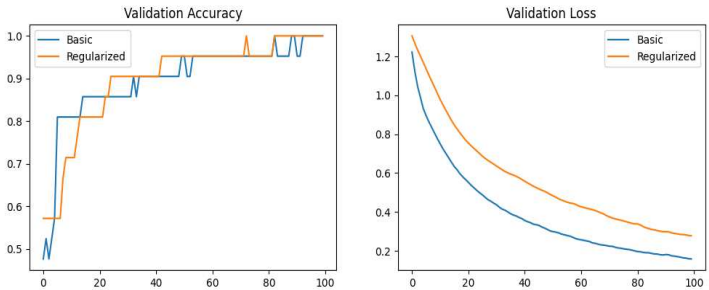
```
Model Basic (SGD) Accuracy:
[0.27128055691719055, 0.8666666746139526]

Model Regularized (Adam) Accuracy:
[0.36375537514686584, 0.9111111164093018]
```

```
print(model_reg.evaluate(X_test, y_test, verbose=0))
```

6. Visualisasi Perbandingan Training

```
plt.figure(figsize=(12,4))
```

```
# Accuracy
plt.subplot(1,2,1)
plt.plot(history_basic.history['val_accuracy'], label='Basic')
plt.plot(history_reg.history['val_accuracy'],
label='Regularized')
plt.title("Validation Accuracy")
plt.legend()

# Loss
    plt.subplot(1,2,2)
    plt.plot(history_basic.history['val_loss'], label='Basic')
    plt.plot(history_reg.history['val_loss'], label='Regularized')
    plt.title("Validation Loss")
```




```
    plt.legend()
    plt.show()
```

DAFTAR PUSTAKA

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

[2] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 37, pp. 132–156, 2017, doi: https://doi.org/10.1016/j.inffus.2017.02.004.

[3] I. Express and L. Part, "AND K-MEANS CLUSTERING," vol. 14, no. 1, pp. 21–28, 2023, doi: 10.24507/icicelb.14.01.21.

[4] P. Mahajan, S. Uddin, F. Hajati, and M. A. Moni, "Ensemble Learning for Disease Prediction: A Review," *Healthcare*, vol. 11, no. 12, 2023, doi: 10.3390/healthcare11121808.

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

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

[7] H. Xie, S. Chen, C. Lai, G. Ma, and W. Huang, "Forecasting the clearing price in the day-ahead spot market using eXtreme Gradient Boosting," *Electr. Eng.*, vol. 104, no. 3, pp. 1607–1621, 2022, doi: 10.1007/s00202-021-01410-6.

[8] A. Mulyana, Y. Hermawan, and I. Mulia, "MACHINE LEARNING," *Kesatuan Press*, vol. 13, no. 1, 2025, [Online]. Available: https://jurnal.ibik.ac.id/index.php/kpress/article/view/3133

[9] M. A. Talukder, M. M. Islam, M. A. Uddin, A. Akhter, K. F. Hasan, and M. A. Moni, "Machine learning-based lung and colon cancer detection using deep feature extraction and ensemble learning," *Expert Syst. Appl.*, vol. 205, p. 117695, 2022, doi: https://doi.org/10.1016/j.eswa.2022.117695.

[10] I. Riadi, R. Umar, and R. Anggara, "Comparative Analysis of Naive Bayes and K-NN Approaches to Predict Timely

Graduation using Academic History," *Int. J. Comput. Digit. Syst.*, vol. 16, no. 1, pp. 1163–1174, 2024, doi: 10.12785/ijcds/160185.

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

[12] L. Munkhdalai, T. Munkhdalai, O.-E. Namsrai, J. Y. Lee, and K. H. Ryu, "An Empirical Comparison of Machine-Learning Methods on Bank Client Credit Assessments," *Sustainability*, vol. 11, no. 3, 2019, doi: 10.3390/su11030699.

[13] A. Panarese, G. Settanni, V. Vitti, and A. Galiano, "Developing and Preliminary Testing of a Machine Learning-Based Platform for Sales Forecasting Using a Gradient Boosting Approach," *Appl. Sci.*, vol. 12, no. 21, 2022, doi: 10.3390/app122111054.

[14] R. Muniappan *et al.*, "An optimized deep learning framework based on LEE for real time student performance prediction in educational data," *Bull. Electr. Eng. Informatics*, vol. 14, no. 5, pp. 3671–3682, 2025, doi: 10.11591/eei.v14i5.9773.

[15] A. Anggrawan, H. Hairani, and C. Satria, "Improving SVM Classification Performance on Unbalanced Student Graduation Time Data Using SMOTE," *Int. J. Inf. Educ. Technol.*, vol. 13, no. 2, pp. 289–295, 2023, doi: 10.18178/ijiet.2023.13.2.1806.

[16] S. Lina, M. Sitio, and N. Rofiq, "Classification of Creditworthy Customer Using Support Vector Machine Algorithm," vol. 10, no. 2, pp. 339–345, 2025, doi: 10.31572/inotera.Vol10.Iss2.2025.ID502.

[17] A. Yaqin, M. Rahardi, and F. F. Abdulloh, "Accuracy Enhancement of Prediction Method using SMOTE for Early Prediction Student's Graduation in XYZ University," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 6, pp. 418–424, 2022, doi: 10.14569/IJACSA.2022.0130652.

[18] A. A. Qureshi, M. Ahmad, S. Ullah, M. N. Yasir, F. Rustam, and I. Ashraf, "Performance evaluation of machine learning models on large dataset of android applications reviews," *Multimed. Tools Appl.*, vol. 82, no. 24, pp. 37197–37219, 2023, doi: 10.1007/s11042-023-14713-6.

[19] M. Hadwan, M. Al-Sarem, F. Saeed, and M. A. Al-Hagery, "An Improved Sentiment Classification Approach for Measuring User Satisfaction toward Governmental Services' Mobile Apps Using Machine Learning Methods with Feature Engineering and SMOTE Technique," *Appl. Sci.*, vol. 12, no. 11, 2022, doi: 10.3390/app12115547.

[20] F. Pedregosa, R. Weiss, and M. Brucher, "Scikit-learn : Machine Learning in Python," vol. 12, pp. 2825–2830, 2011.

[21] Jan Melvin Ayu Soraya Dachi and Pardomuan Sitompul, "Analisis Perbandingan Algoritma XGBoost dan Algoritma Random Forest Ensemble Learning pada Klasifikasi Keputusan Kredit," *J. Ris. Rumpun Mat. Dan Ilmu Pengetah. Alam*, vol. 2, no. 2, pp. 87–103, 2023, doi: 10.55606/jurrimipa.v2i2.1470.

[22] M. Sivakumar, S. Parthasarathy, and T. Padmapriya, "Trade-off between training and testing ratio in machine learning for medical image processing.," *PeerJ. Comput. Sci.*, vol. 10, p. e2245, 2024, doi: 10.7717/peerj-cs.2245.

[23] S. Zhao, D. Zhou, H. Wang, D. Chen, and L. Yu, "Enhancing Student Academic Success Prediction Through Ensemble Learning and Image-Based Behavioral Data Transformation," *Appl. Sci.*, vol. 15, no. 3, 2025, doi: 10.3390/app15031231.

[24] E. M. Ferrouhi and I. Bouabdallaoui, "A comparative study of ensemble learning algorithms for high-frequency trading," *Sci. African*, vol. 24, p. e02161, 2024, doi: https://doi.org/10.1016/j.sciaf.2024.e02161.

[25] K. Okoye, J. T. Nganji, J. Escamilla, and S. Hosseini, "Machine learning model (RG-DMML) and ensemble algorithm for prediction of students' retention and graduation in education," *Comput. Educ. Artif. Intell.*, vol. 6, no. January, p. 100205, 2024, doi: 10.1016/j.caeai.2024.100205.

[26] Y. Rimal and N. Sharma, "Ensemble machine learning prediction accuracy: local vs. global precision and recall for multiclass grade performance of engineering students," *Front. Educ.*, vol. 10, no. April, pp. 1–16, 2025, doi: 10.3389/feduc.2025.1571133.

[27] S. Kim, E. Choi, Y.-K. Jun, and S. Lee, "Student Dropout

Prediction for University with High Precision and Recall," *Appl. Sci.*, vol. 13, no. 10, 2023, doi: 10.3390/app13106275.

[28] P. D. Rinanda and Mustakim, "Implementation of PNN, ANN And K-NN Algorithms on Indonesian Marketplace Reviews on Google Play Store," in *2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS)*, 2024, pp. 1070–1074. doi: 10.1109/ICETSIS61505.2024.10459477.

[29] P. Yan, W. Chu, and H. Wu, "A Multimodal Deep Learning Model for Optimizing Music Emotion Recognition Through Temporal and Semantic Feature Integration," *J. Circuits, Syst. Comput.*, vol. 0, no. ja, p. null, doi: 10.1142/S0218126626500581.

[30] R. Pahuja and A. Kumar, "Sound-spectrogram based automatic bird species recognition using MLP classifier," *Appl. Acoust.*, vol. 180, p. 108077, 2021, doi: https://doi.org/10.1016/j.apacoust.2021.108077.

[31] L. Huang *et al.*, "Combining Random Forest and XGBoost Methods in Detecting Early and Mid-Term Winter Wheat Stripe Rust Using Canopy Level Hyperspectral Measurements," *Agriculture*, vol. 12, no. 1, 2022, doi: 10.3390/agriculture12010074.

[32] F. Niklas, E. Annac, and A. Wirth, "App-based learning for kindergarten children at home (Learning4Kids): study protocol for cohort 1 and the kindergarten assessments," *BMC Pediatr.*, vol. 20, no. 1, p. 554, 2020, doi: 10.1186/s12887-020-02432-y.

TENTANG PENULIS

Endar Nirmala, S.Kom.,M.T. Lahir di Jakarta, 24 Januari 1967. Saya menempuh Magister Teknik dengan Konsentrasi Teknologi Informasi di Institut Teknologi Bandung Tahun 2006. Sejak tahun 2008 saya menjadi dosen di Prodi Teknik Informatika, Universitas Pamulang. Penelitian yang saya buat berfokus pada Software Engineering, Sistem Informasi dan Data Science.

Buku **"Metode Klasifikasi Modern dalam Machine Learning Menggunakan Python"** membahas secara komprehensif konsep, metode, dan implementasi algoritma klasifikasi yang banyak digunakan dalam bidang Machine Learning. Buku ini dimulai dengan pengenalan konsep dasar Machine Learning, termasuk jenis pembelajaran seperti supervised, unsupervised, dan reinforcement learning, serta tahapan umum dalam proyek Machine Learning mulai dari pengumpulan data, preprocessing, hingga evaluasi model. Pembaca juga diperkenalkan dengan berbagai teknik pengolahan data seperti data cleaning, encoding data kategorikal, normalisasi, standardisasi, serta teknik feature selection dan feature engineering yang penting untuk meningkatkan kualitas model. Selain itu, buku ini memberikan pemahaman konseptual yang kuat mengenai berbagai algoritma klasifikasi seperti Support Vector Machine (SVM), Decision Tree, Random Forest, dan Artificial Neural Network (ANN).

Perkembangan teknologi informasi dan data dalam beberapa dekade terakhir telah melahirkan berbagai inovasi dalam bidang kecerdasan buatan (artificial intelligence) dan machine learning. Salah satu teknik yang sangat penting dalam machine learning adalah metode klasifikasi, yaitu proses pengelompokan data ke dalam kategori tertentu berdasarkan pola atau karakteristik yang dimiliki oleh data tersebut. Metode ini banyak digunakan dalam berbagai bidang, seperti analisis data bisnis, pengenalan wajah, diagnosis medis, sistem rekomendasi, analisis sentimen, hingga keamanan siber.

Buku ini dirancang untuk memberikan pemahaman yang sistematis mengenai konsep dasar machine learning, khususnya metode klasifikasi modern. Pembahasan dalam buku ini mencakup berbagai algoritma klasifikasi yang banyak digunakan, seperti Decision Tree, Naïve Bayes, Support Vector Machine, K-Nearest Neighbor, hingga metode berbasis ensemble learning. Selain itu, buku ini juga menyajikan contoh implementasi praktis menggunakan Python dengan berbagai pustaka populer seperti NumPy, Pandas, Scikit-learn, dan Matplotlib, sehingga pembaca dapat memahami tidak hanya konsep teoretis, tetapi juga penerapannya dalam analisis data nyata.

MEDIA PUSTAKA INDO

PT MEDIA PUSTAKA INDO
Jl. Merdeka RT4/RW2
Binangun, Kab. Cilacap, Provinsi Jawa Tengah
No hp. 0838 6333 3823
Website: www.mediapustakaindo.com
E-mail: mediapustakaindo@gmail.com

ISBN 978-634-247-279-8
9 786342 472798