Machine Learning Cheatsheet *: Model Klasifikasi*

Disusun oleh Tim Datasans

https://datasans.medium.com/

Table of Contents

Bab 1: Pendahuluan: Mengenal Dunia Klasifikasi dalam Machine Learning

1.1. Pengantar

Machine Learning (ML) merupakan salah satu cabang ilmu dalam bidang kecerdasan buatan (Artificial Intelligence, AI) yang berkembang pesat dalam beberapa tahun terakhir. ML memungkinkan komputer untuk "belajar" dari data dan membuat prediksi atau mengambil keputusan tanpa diprogram secara eksplisit. Salah satu tugas utama dalam machine learning adalah klasifikasi, yang merupakan fokus utama dalam buku ini.

Klasifikasi adalah proses pengelompokan objek atau data ke dalam beberapa kelas atau kategori berdasarkan fitur-fitur yang dimilikinya. Pada bab ini, kita akan mempelajari konsep dasar klasifikasi dalam machine learning, jenis-jenis klasifikasi, dan beberapa terminologi penting yang akan sering kita temui sepanjang buku ini.

1.2. Klasifikasi dalam Machine Learning

Dalam konteks machine learning, klasifikasi adalah tugas supervised learning di mana model ML dibentuk berdasarkan data latih yang sudah diberi label kelas. Dalam kata lain, model ML harus "belajar" dari data latih untuk kemudian dapat mengklasifikasikan data baru yang belum memiliki label kelas. Beberapa contoh aplikasi klasifikasi dalam dunia nyata adalah:

Deteksi spam: membedakan antara email yang merupakan spam dan yang bukan spam.
Diagnostik medis: mengidentifikasi penyakit berdasarkan gejala atau hasil tes.
Pengenalan karakter tulisan tangan: mengklasifikasikan karakter tulisan tangan menjadi angka atau huruf yang sesuai.
Analisis sentimen: menilai apakah suatu teks memiliki sentimen positif, negatif, atau netral.
Kredit skoring: memprediksi apakah seseorang akan mengembalikan pinjaman atau tidak.

1.3. Jenis-jenis Klasifikasi

Klasifikasi dalam machine learning umumnya dibagi menjadi dua jenis, yaitu:

Klasifikasi Binomial (Binary Classification): Klasifikasi ini hanya melibatkan dua kelas atau kategori. Contohnya adalah deteksi spam yang membedakan antara email spam dan bukan spam.

Klasifikasi Multiclass (Multiclass Classification): Klasifikasi ini melibatkan lebih dari dua kelas atau kategori. Contohnya adalah pengenalan karakter tulisan tangan yang melibatkan banyak kelas, yaitu angka 0-9 dan huruf A-Z.

1.4. Terminologi Penting dalam Klasifikasi

Berikut adalah beberapa istilah penting yang perlu kita ketahui dalam klasifikasi machine learning:

Fitur (Feature)

Fitur adalah variabel input yang digunakan untuk menggambarkan objek atau data yang akan diklasifikasikan. Fitur dapat berupa numerik, kategorikal, atau bahkan teks. Sebagai contoh, dalam diagnostik medis, fitur bisa meliputi usia, jenis kelamin, dan hasil tes darah.

Label (Label)

Label adalah variabel target yang menjadi output dalam klasifikasi. Label menggambarkan kelas atau kategori yang sebenarnya dari objek atau data. Misalnya, dalam deteksi spam, label bisa berupa "spam" atau "bukan spam."

Data Latih (Training Data)

Data latih adalah kumpulan data yang digunakan untuk melatih model ML. Data latih terdiri dari fitur dan label yang sudah diketahui. Model ML akan "belajar" dari data latih untuk mengidentifikasi pola dan hubungan antara fitur dan label.

Data Uji (Testing Data)

Data uji adalah kumpulan data yang digunakan untuk menguji kinerja model ML setelah proses pelatihan. Data uji terdiri dari fitur tetapi tidak menyertakan label. Model ML akan membuat prediksi label untuk data uji, yang kemudian dapat dibandingkan dengan label sebenarnya (jika diketahui) untuk menghitung metrik evaluasi.

Overfitting

Overfitting terjadi ketika model ML terlalu baik dalam mempelajari pola dan detail dari data latih, sehingga menjadi kurang mampu untuk membuat prediksi yang akurat pada data baru yang belum pernah dilihat sebelumnya. Overfitting biasanya terjadi ketika model ML terlalu kompleks dan mencoba untuk menyesuaikan diri dengan setiap titik data dalam data latih.

Underfitting

Underfitting terjadi ketika model ML tidak mampu menangkap pola dan hubungan yang ada dalam data latih, sehingga menghasilkan prediksi yang buruk baik pada data latih maupun data uji. Underfitting biasanya terjadi ketika model ML terlalu sederhana dan tidak memiliki kapasitas yang cukup untuk mempelajari pola yang ada dalam data latih.

Regularisasi

Regularisasi adalah teknik yang digunakan untuk mengurangi overfitting dalam model ML dengan menambahkan hukuman (penalty) terhadap kompleksitas model. Regularisasi mendorong model untuk menjadi lebih sederhana dan menghindari penyesuaian yang berlebihan terhadap data latih.

Cross-Validation

Cross-validation adalah teknik yang digunakan untuk mengestimasi kinerja model ML pada data baru yang belum pernah dilihat sebelumnya. Dalam cross-validation, data latih dibagi menjadi beberapa lipatan (folds), di mana model ML dilatih pada sebagian lipatan dan diuji pada lipatan yang tersisa. Proses ini diulangi beberapa kali dengan lipatan yang berbeda, dan hasilnya dirata-ratakan untuk menghasilkan estimasi kinerja yang lebih stabil dan dapat diandalkan.

Rangkuman

Dalam bab-bab berikutnya, kita akan mempelajari berbagai algoritma klasifikasi yang populer, mulai dari regresi logistik hingga metode gabungan (ensemble methods), serta cara mengevaluasi dan menginterpretasi kinerja model klasifikasi. Tujuan utama buku ini adalah untuk membantu Anda memahami konsep dasar dan teknik-teknik klasifikasi dalam machine learning, sehingga Anda dapat mengaplikasikannya dalam berbagai masalah dunia nyata.

Bab 2: Regresi Logistik: Memahami Dasar Klasifikasi Binomial

2.1 Pendahuluan

Regresi logistik merupakan salah satu algoritma klasifikasi paling sederhana dan umum digunakan untuk klasifikasi binomial. Algoritma ini menggunakan fungsi logistik untuk memprediksi probabilitas suatu objek termasuk dalam satu kelas atau kelas lain.

2.2 Algoritma Regresi Logistik

Regresi logistik menggunakan fungsi logistik (sigmoid) untuk memprediksi probabilitas suatu objek termasuk dalam satu kelas atau kelas lain. Fungsi logistik memiliki bentuk berikut:

```
P(y=1|x) = 1 / (1 + exp(-(w0 + w1 * x1 + w2 * x2 + ... + wn * xn)))
```

Di mana:

P(y=1|x) adalah probabilitas objek termasuk dalam kelas 1, dengan x sebagai fitur objek.
w0, w1, ..., wn adalah koefisien regresi yang harus dipelajari dari data.
exp() adalah fungsi eksponensial.

2.3 Matematika di Belakang Regresi Logistik

Regresi logistik menggunakan metode Maximum Likelihood Estimation (MLE) untuk mempelajari koefisien regresi. MLE mencoba memaksimalkan kemungkinan data dengan memilih koefisien yang paling mungkin menghasilkan data yang diamati.

2.4 Kasus-Kasus yang Direkomendasikan

Regresi logistik direkomendasikan untuk kasus-kasus berikut:

1. Klasifikasi binomial dengan fitur numerik atau kategorikal.
2. Hubungan antara variabel prediktor dan target bersifat linear.
3. Masalah yang memerlukan interpretasi koefisien yang mudah.

4. Sebagai baseline untuk algoritma klasifikasi yang lebih kompleks.

2.5 Contoh Kasus di Dunia Nyata dan Python Script

Misalkan kita memiliki data sintetis tentang siswa dan apakah mereka lulus ujian atau tidak. Fitur yang kita miliki adalah waktu belajar (jam) dan skor praujian.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Membuat data sintetis
data = {'Waktu_Belajar': [5, 8, 6, 4, 7, 10, 9, 3, 2, 11],
        'Skor_Praujian': [60, 80, 70, 50, 75, 85, 90, 40, 30, 95],
        'Lulus': [0, 1, 1, 0, 1, 1, 1, 0, 0, 1]}

df = pd.DataFrame(data)

# Membagi data menjadi fitur (X) dan target (y)
X = df.drop(columns=['Lulus'])
y = df['Lulus']

# Membagi data menjadi data pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Membuat model Regresi Logistik
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Memprediksi data pengujian
y_pred = log_reg.predict(X_test)

# Menghitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi:", accuracy)
```

2.6 Penjelasan Parameter untuk Hyperparameter Tuning

Parameter utama yang perlu diperhatikan dalam regresi logistik adalah:
- `penalty`: Jenis regularisasi yang digunakan ('l1', 'l2', 'elasticnet', atau 'none').

- `C`: Nilai kebalikan dari kekuatan regularisasi (nilai yang lebih kecil menyebabkan regularisasi yang lebih kuat).
- `fit_intercept`: Jika True, maka akan mencakup intersep dalam model (biasa disebut w0).
- `solver`: Algoritma yang digunakan untuk mengoptimalkan koefisien (misal 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga').

2.7 Hal-Hal yang Harus Diperhatikan

- Periksa asumsi linearitas antara fitur dan logit dari variabel target.
- Transformasi fitur yang memiliki distribusi yang sangat tidak simetris atau memiliki outliers.
- Regularisasi bisa digunakan untuk mengurangi overfitting dan meningkatkan generalisasi.

2.8 Kelebihan dan Kekurangan Regresi Logistik

Kelebihan	Kekurangan
Mudah diinterpretasi dan diimplementasikan	Terbatas pada asumsi linearitas antara fitur dan logit dari variabel target
Parameter yang bisa dituning	Kurang efisien untuk klasifikasi multi-kelas (memerlukan strategi seperti one-vs-rest atau one-vs-one)
Menghasilkan probabilitas yang berguna untuk analisis	Kurang efektif untuk fitur kategorikal dengan banyak kategori atau ketika hubungan antara fitur dan target bersifat non-linear
Cepat dalam proses pelatihan dan prediksi	Memerlukan pemrosesan data tambahan seperti transformasi fitur yang memiliki distribusi tidak simetris atau mengandung pencilan

2.9 Regresi Logistik untuk Klasifikasi Multi-Kelas

Meskipun regresi logistik secara alami cocok untuk klasifikasi binomial, kita juga bisa menggunakannya untuk klasifikasi multi-kelas dengan menggunakan strategi one-vs-rest atau one-vs-one. Berikut adalah contoh cara menggunakannya untuk klasifikasi multi-kelas dengan menggunakan dataset Iris:

```
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

# Memuat dataset Iris
iris = load_iris()
X = iris.data
y = iris.target

# Membagi data menjadi data pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Membuat model Regresi Logistik dengan strategi multi-kelas
'multinomial'
log_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs',
max_iter=1000)
log_reg.fit(X_train, y_train)

# Memprediksi data pengujian
y_pred = log_reg.predict(X_test)

# Menghitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi:", accuracy)
```

2.10 Kesimpulan

Regresi logistik merupakan algoritma klasifikasi yang sederhana dan mudah diinterpretasikan, cocok untuk klasifikasi binomial dengan fitur numerik atau kategorikal. Meskipun memiliki beberapa kelemahan, seperti asumsi linearitas dan keterbatasan untuk klasifikasi multi-kelas, regresi logistik tetap menjadi pilihan yang baik untuk banyak masalah klasifikasi, terutama sebagai baseline untuk algoritma yang lebih kompleks.

Hyperparameter tuning, transformasi fitur, dan penggunaan strategi multi-kelas bisa membantu meningkatkan performa model regresi logistik.

Bab 3: K-Nearest Neighbors: Algoritma Berbasis Jarak untuk Klasifikasi

3.1 Penjelasan Algoritma

K-Nearest Neighbors (KNN) adalah algoritma pembelajaran mesin yang digunakan untuk masalah klasifikasi dan regresi. Algoritma ini bekerja berdasarkan prinsip "kemiripan". Dalam konteks klasifikasi, objek baru akan diklasifikasikan berdasarkan mayoritas kelas dari k-tetangga terdekatnya.

3.2 Matematika di Belakangnya

KNN menggunakan jarak antara poin data untuk menentukan tetangga terdekat. Beberapa metrik jarak yang umum digunakan adalah:

Jarak Euclidean: Jarak garis lurus antara dua titik dalam ruang Euclidean.
Jarak Manhattan: Jarak yang diukur melalui jalur grid, bukan garis lurus.
Jarak Minkowski: Generalisasi dari jarak Euclidean dan Manhattan.
Rumus jarak Euclidean:

$$d(p, q) = \sqrt{[(p1-q1)^2 + (p2-q2)^2 + \ldots + (pn-qn)^2]}$$

3.3 Kasus yang Direkomendasikan

KNN cocok digunakan dalam kasus berikut:

1. Data dengan jumlah fitur yang relatif kecil.
2. Masalah klasifikasi non-linear.
3. Ketika tidak ada asumsi awal tentang distribusi data.

3.4 Contoh Kasus di Dunia Nyata dan Python Script

Contoh: Klasifikasi jenis kelamin berdasarkan tinggi dan berat badan.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

```
# Membuat data sintetis
data = {'Tinggi': [170, 167, 159, 155, 162, 175, 168, 182, 177, 190],
        'Berat': [65, 62, 54, 48, 52, 72, 68, 80, 78, 90],
        'Jenis_Kelamin': ['L', 'L', 'P', 'P', 'P', 'L', 'L', 'L', 'L',
'L']}

df = pd.DataFrame(data)

# Membagi data menjadi fitur (X) dan target (y)
X = df.drop(columns=['Jenis_Kelamin'])
y = df['Jenis_Kelamin']

# Membagi data menjadi data pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Membuat model KNN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Memprediksi data pengujian
y_pred = knn.predict(X_test)

# Menghitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi:", accuracy)
```

3.5 Penjelasan Setiap Parameter untuk Hyperparameter Tuning

1. n_neighbors: Jumlah tetangga yang akan digunakan dalam proses klasifikasi (default: 5).
2. weights: Fungsi pembobotan untuk tetangga (uniform, distance, atau callable(custom)). Default adalah 'uniform', yang berarti semua tetangga memiliki bobot yang sama. 'Distance' memberikan bobot berdasarkan jarak (semakin dekat, semakin berpengaruh).
3. algorithm: Algoritma yang digunakan untuk menghitung tetangga terdekat (auto, ball_tree, kd_tree, atau brute). Default adalah 'auto', yang berarti algoritma akan dipilih secara otomatis.
4. leaf_size: Ukuran leaf yang digunakan jika algoritma adalah BallTree atau KDTree. Leaf lebih besar mengurangi waktu pencarian tetapi membutuhkan lebih banyak memori.

5. p: Parameter kekuatan untuk metrik jarak Minkowski. p=1 adalah jarak Manhattan, p=2 adalah jarak Euclidean, dan untuk p>2 akan menjadi jarak Minkowski.
6. metric: Metrik jarak yang digunakan untuk menghitung jarak antara poin data (default: 'minkowski').
7. n_jobs: Jumlah prosesor paralel yang digunakan untuk menghitung tetangga terdekat. Default adalah None, yang berarti hanya menggunakan 1 prosesor.

3.6 Hal-Hal yang Harus Diperhatikan

Penting untuk menormalisasi fitur sebelum menggunakan KNN, karena fitur dengan rentang yang lebih besar akan memiliki pengaruh yang lebih besar pada jarak antar poin data.
KNN sensitif terhadap jumlah tetangga (k) yang digunakan. Nilai k yang terlalu kecil bisa menyebabkan overfitting, sedangkan k yang terlalu besar bisa menyebabkan underfitting. Gunakan metode seperti cross-validation untuk menemukan nilai k yang optimal.
KNN cenderung lambat untuk data dengan jumlah fitur yang besar, karena perhitungan jarak menjadi lebih kompleks.

3.7 Pros dan Cons

Pros	Cons
Mudah dipahami dan diimplementasikan	Sensitif terhadap jumlah tetangga (k)
Tidak perlu melatih model	Rentan terhadap data *noise*
Cocok untuk data non-linear	Skalabilitas rendah untuk data berdimensi tinggi
Tidak memerlukan asumsi awal tentang distribusi data	Waktu komputasi yang lama untuk dataset besar

3.8 Optimasi KNN dengan Cross-Validation

Untuk menemukan nilai k yang optimal, kita bisa menggunakan metode cross-validation. Berikut contoh cara melakukannya dengan menggunakan library scikit-learn:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Membuat data sintetis
data = {'Tinggi': [170, 167, 159, 155, 162, 175, 168, 182, 177, 190],
        'Berat': [65, 62, 54, 48, 52, 72, 68, 80, 78, 90],
        'Jenis_Kelamin': ['L', 'L', 'P', 'P', 'P', 'L', 'L', 'L', 'L',
'L']}

df = pd.DataFrame(data)

# Membagi data menjadi fitur (X) dan target (y)
X = df.drop(columns=['Jenis_Kelamin'])
y = df['Jenis_Kelamin']

# Membagi data menjadi data pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Membuat model KNN
knn = KNeighborsClassifier()

# Menyiapkan parameter untuk tuning
params = {'n_neighbors': list(range(1, 6)),
          'weights': ['uniform', 'distance'],
          'metric': ['euclidean', 'manhattan', 'minkowski']}

# Melakukan optimasi dengan GridSearchCV
grid_search = GridSearchCV(knn, params, cv=5, scoring='accuracy',
n_jobs=-1)
grid_search.fit(X_train, y_train)

# Menampilkan parameter terbaik
print("Parameter Terbaik:", grid_search.best_params_)

# Memprediksi data pengujian
y_pred = grid_search.predict(X_test)

# Menghitung akurasi
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi:", accuracy)
```

Dengan menggunakan GridSearchCV, kita bisa mencari nilai k dan parameter lain yang optimal untuk meningkatkan performa model KNN. Selalu penting untuk melakukan tuning dan validasi model untuk menghindari overfitting atau underfitting.

3.9 Kasus Khusus: Klasifikasi Multi-Kelas

KNN juga bisa digunakan untuk klasifikasi multi-kelas. Dalam hal ini, kelas objek baru akan ditentukan berdasarkan mayoritas kelas dari k-tetangga terdekatnya. Berikut contoh cara mengimplementasikannya:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Memuat dataset Iris
iris = load_iris()
X = iris.data
y = iris.target

# Membagi data menjadi data pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Membuat model KNN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Memprediksi data pengujian
y_pred = knn.predict(X_test)

# Menghitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi:", accuracy)
```

3.10 Kesimpulan

KNN merupakan algoritma yang mudah dipahami dan diimplementasikan. Cocok untuk masalah klasifikasi dengan data non-linear dan jumlah fitur yang kecil. Namun, KNN

memiliki beberapa kelemahan, seperti sensitivitas terhadap jumlah tetangga, rentan terhadap data berisik, dan waktu komputasi yang lama untuk dataset besar.

Untuk meningkatkan performa model KNN, penting untuk menormalisasi fitur, menemukan nilai k yang optimal, serta melakukan tuning parameter menggunakan metode seperti cross-validation. Dalam beberapa kasus, menggabungkan KNN dengan metode ensemble seperti bagging atau boosting bisa meningkatkan akurasi dan keandalan model.

Bab 4: Support Vector Machines: Membangun Batas Keputusan dengan Hyperplane

4.1 Pendahuluan

Support Vector Machines (SVM) merupakan algoritma klasifikasi yang kuat dan fleksibel, yang bekerja dengan mencari hyperplane terbaik yang memisahkan data menjadi dua kelas atau lebih. SVM sangat efektif dalam mengatasi masalah klasifikasi linear dan non-linear.

4.2 Algoritma Support Vector Machines

SVM mencari hyperplane yang memaksimalkan margin antara kelas. Margin didefinisikan sebagai jarak terkecil antara hyperplane dan titik-titik data terdekat dari kedua kelas. Titik-titik data ini disebut support vectors.

4.3 Matematika di Belakang Support Vector Machines

Untuk kasus linear, SVM mencari hyperplane yang memenuhi persamaan:

$$w * x + b = 0$$

Di mana w adalah vektor berat, x adalah vektor fitur, dan b adalah bias. SVM meminimalkan ||w|| sambil mempertahankan margin yang lebih besar atau sama dengan 1:

$$y(i) * (w * x(i) + b) >= 1$$

Untuk kasus non-linear, SVM menggunakan kernel trick untuk memetakan data ke dimensi yang lebih tinggi, di mana data bisa dipisahkan secara linear. Kernel yang umum digunakan adalah polynomial, Gaussian Radial Basis Function (RBF), dan sigmoid.

4.4 Kasus-Kasus yang Direkomendasikan

SVM direkomendasikan untuk kasus-kasus berikut:

1. Klasifikasi binomial atau multi-kelas.

2. Data dengan fitur numerik atau kategorikal (yang telah dikonversi menjadi numerik).
3. Masalah dengan pola yang kompleks dan hubungan non-linear antara fitur dan target.

4.5 Contoh Kasus di Dunia Nyata dan Python Script

Misalkan kita memiliki data sintetis tentang penggunaan mesin ATM dan apakah transaksi berhasil atau gagal. Fitur yang kita miliki adalah saldo awal (dalam dolar) dan jumlah penarikan (dolar).

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Membuat data sintetis
data = {'Saldo_Awal': [1000, 500, 2000, 1500, 1200, 800, 3000, 2500,
4000, 3500],
        'Penarikan': [100, 200, 300, 400, 500, 600, 700, 800, 900,
1000],
        'Transaksi_Berhasil': [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]}

df = pd.DataFrame(data)

# Membagi data menjadi fitur (X) dan target (y)
X = df.drop(columns=['Transaksi_Berhasil'])
y = df['Transaksi_Berhasil']

# Membagi data menjadi data pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Membuat model Support Vector Machines
svm = SVC(kernel='rbf')
svm.fit(X_train, y_train)

# Memprediksi data pengujian
y_pred = svm.predict(X_test)

# Menghitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi:", accuracy)
```

4.6 Penjelasan Parameter untuk Hyperparameter Tuning

Parameter utama yang perlu diperhatikan dalam SVM adalah:
- `C`: Nilai yang mengontrol trade-off antara margin yang lebih besar dan jumlah kesalahan klasifikasi (nilai yang lebih kecil menyebabkan margin yang lebih besar).
- `kernel`: Fungsi kernel yang digunakan ('linear', 'poly', 'rbf', 'sigmoid', atau custom).
- `degree`: Derajat polynomial untuk kernel polynomial (hanya berlaku jika `kernel='poly'`).
- `gamma`: Koefisien untuk kernel RBF, polynomial, dan sigmoid (jika `gamma='scale'`, maka dihitung sebagai 1 / (n_features * X.var())).
- `coef0`: Istilah independen dalam kernel polynomial dan sigmoid.

4.7 Hal-Hal yang Harus Diperhatikan

- Normalisasi data sangat penting untuk meningkatkan performa SVM.
- SVM sangat sensitif terhadap hyperparameter, jadi penggunaan cross-validation sangat dianjurkan.
- Pemilihan kernel yang tepat sangat penting, karena kernel yang salah bisa menyebabkan performa yang buruk.

4.8 Kelebihan dan Kekurangan Support Vector Machines

Kelebihan	Kekurangan
Efektif dalam klasifikasi linear dan non-linear	Memerlukan normalisasi data
Fleksibel dengan berbagai pilihan kernel	Sering memerlukan tuning hyperparameter yang cermat

bisa menghasilkan model yang akurat	Kurang mudah diinterpretasi dibandingkan dengan model lain
Mengendalikan risiko overfitting dengan mengatur margin	Kurang efisien untuk dataset yang sangat besar

Bab 5: Decision Trees: Klasifikasi Berbasis Struktur Pohon Keputusan

5.1 Pendahuluan

Decision Trees (Pohon Keputusan) merupakan metode klasifikasi yang populer dan mudah diinterpretasi. Mereka membangun struktur pohon yang berdasarkan pada fitur dan kondisi yang dipilih untuk memecah dataset menjadi kelas yang berbeda.

5.2 Algoritma Decision Trees

Algoritma pohon keputusan bekerja dengan memilih fitur terbaik untuk memecah data dan mengulanginya secara rekursif hingga mencapai kondisi berhenti, seperti kedalaman maksimum pohon atau jumlah minimum sampel yang diperlukan untuk memecah simpul.

5.3 Matematika di Belakang Decision Trees

Metode umum untuk memilih fitur terbaik adalah dengan mengukur impurity dari simpul menggunakan metrik seperti Gini impurity atau Entropy. Impurity yang lebih rendah mengindikasikan simpul yang lebih "murni" atau lebih baik dalam mengklasifikasikan data.

5.4 Kasus yang Direkomendasikan untuk Menggunakan Decision Trees

Decision Trees sangat cocok untuk:

1. Klasifikasi data yang memiliki fitur kategorikal atau numerik
2. Masalah yang memerlukan interpretasi yang mudah dan bisa dijelaskan
3. Ketika overfitting bisa dikendalikan dengan membatasi kedalaman pohon atau prunning

5.5 Contoh Kasus di Dunia Nyata dan Python Script

Contoh: Mengklasifikasikan jenis kelamin berdasarkan tinggi, berat, dan ukuran sepatu.

```
import numpy as np
```

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

# Data sintetis
X = np.array([[180, 80, 44], [177, 70, 43], [160, 60, 38], [154, 54,
37], [166, 65, 40], [190, 90, 47], [175, 64, 39], [177, 70, 40], [159,
55, 37], [171, 75, 42], [181, 85, 43]])
Y = np.array(['pria', 'pria', 'wanita', 'wanita', 'pria', 'pria',
'wanita', 'wanita', 'wanita', 'pria', 'pria'])

# Membagi data menjadi data latih dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
random_state=42)

# Membuat model Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

# Memprediksi data uji
y_pred = dt.predict(X_test)

# Menghitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi:", accuracy)
```

5.6 Penjelasan Parameter untuk Hyperparameter Tuning

criterion: Fungsi untuk mengukur kualitas pemisahan simpul ('gini' atau 'entropy')
max_depth: Kedalaman maksimum pohon
min_samples_split: Jumlah minimum sampel yang diperlukan untuk memecah simpul internal
min_samples_leaf: Jumlah minimum sampel yangdiperlukan di simpul daun
max_features: Jumlah fitur yang akan dipertimbangkan saat mencari pemisahan terbaik
max_leaf_nodes: Batas maksimum simpul daun
min_impurity_decrease: Batas penurunan impurity minimum yang harus dicapai untuk memecah simpul

5.7 Hal-Hal yang Harus Diperhatikan

Pengendalian overfitting dengan membatasi kedalaman pohon atau melakukan pruning
Penting untuk menyeimbangkan pohon agar kinerja lebih optimal

Mengatasi fitur kategorikal yang memiliki banyak kategori dengan menggabungkan atau mengurangi jumlah kategori

5.8 Kelebihan dan Kekurangan Decision Trees

Kelebihan	Kekurangan
Mudah diinterpretasi dan dijelaskan	Rentan terhadap overfitting
Tidak memerlukan normalisasi data	Tidak cocok untuk klasifikasi linier
bisa menangani fitur kategorikal dan numerik	Sensitif terhadap rotasi data
Pemrosesan awal yang minimal	Mungkin tidak menghasilkan pohon yang optimal secara global

Bab 6: Random Forest: Menggabungkan Kekuatan Decision Trees

6.1 Pendahuluan

Random Forest adalah metode klasifikasi yang menggabungkan kekuatan beberapa Decision Trees untuk menghasilkan prediksi yang lebih baik. Dalam proses ini, algoritma mengurangi varians dan overfitting yang mungkin terjadi pada Decision Trees tunggal.

6.2 Algoritma Random Forest

Random Forest bekerja dengan membuat sejumlah Decision Trees dan menggabungkan hasil prediksi mereka melalui metode voting. Setiap pohon dibangun dengan menggunakan bagian acak dari data latih dan fitur, proses ini dikenal sebagai 'bagging' dan 'feature randomness'.

6.3 Matematika di Belakang Random Forest

Random Forest tidak memiliki matematika yang spesifik di luar Decision Trees, karena prinsip dasarnya adalah menggabungkan kekuatan beberapa pohon keputusan.

6.4 Kasus yang Direkomendasikan untuk Menggunakan Random Forest

Random Forest cocok untuk:

1. Klasifikasi data yang memiliki fitur kategorikal atau numerik
2. Masalah yang memerlukan ketahanan terhadap overfitting
3. Ketika kinerja yang lebih baik daripada Decision Trees tunggal diinginkan

6.5 Contoh Kasus di Dunia Nyata dan Python Script

Contoh: Mengklasifikasikan jenis kelamin berdasarkan tinggi, berat, dan ukuran sepatu.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Data sintetis
X = np.array([[180, 80, 44], [177, 70, 43], [160, 60, 38], [154, 54,
37], [166, 65, 40], [190, 90, 47], [175, 64, 39], [177, 70, 40], [159,
55, 37], [171, 75, 42], [181, 85, 43]])
Y = np.array(['pria', 'pria', 'wanita', 'wanita', 'pria', 'pria',
'wanita', 'wanita', 'wanita', 'pria', 'pria'])

# Membagi data menjadi data latih dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
random_state=42)

# Membuat model Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Memprediksi data uji
y_pred = rf.predict(X_test)

# Menghitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi:", accuracy)
```

6.6 Penjelasan Parameter untuk Hyperparameter Tuning

n_estimators: Jumlah Decision Trees dalam Random Forest
criterion: Fungsi untuk mengukur kualitas pemisahan simpul ('gini' atau 'entropy')
max_depth: Kedalaman maksimum pohon
min_samples_split: Jumlah minimum sampel yang diperlukan untuk memecah simpul internal
min_samples_leaf: Jumlah minimum sampel yang diperlukan di simpul daun
max_features: Jumlah fitur yang akan dipertimbangkan saat mencari pemisahan terbaik

max_leaf_nodes: Batas maksimum simpul daun
min_impurity_decrease: Batas penurunan impurity minimum yang harus dicapai untuk memecah simpul
bootstrap: Metode untuk mengambil sampel data latih (True untuk bagging, False untuk pemilihan tanpa penggantian)
oob_score: Menggunakan out-of-bag samples untuk mengestimasi akurasi model

n_jobs: Jumlah pekerjaan paralel yang diizinkan untuk komputasi
random_state: Seed untuk pengulangan yang bisa direproduksi
verbose: Mengontrol keluaran pesan verbositas
warm_start: Menggunakan hasil sebelumnya sebagai inisialisasi untuk model baru

6.7 Hal-Hal yang Harus Diperhatikan

Menyertakan jumlah pohon yang cukup dalam hutan untuk mengurangi varians dan meningkatkan kinerja
Menyetel parameter secara optimal untuk menghindari overfitting atau underfitting
Menyadari waktu pelatihan yang lebih lama dan kebutuhan memori yang lebih tinggi dibandingkan dengan metode lain

6.8 Kelebihan dan Kekurangan Random Forest

Kelebihan	Kekurangan
Mengurangi overfitting	Memerlukan lebih banyak waktu pelatihan
bisa menangani fitur kategorikal dan numerik	Memerlukan lebih banyak memori
Memberikan pentingnya fitur	Sulit diinterpretasi dibandingkan dengan Decision Trees tunggal
Tidak memerlukan normalisasi data	

Bab 7: Naive Bayes: Mengaplikasikan Teorema Bayes dalam Klasifikasi

7.1 Pendahuluan

Naive Bayes adalah algoritma klasifikasi yang didasarkan pada Teorema Bayes, menggabungkan probabilitas prior dan likelihood untuk menghasilkan probabilitas posterior. Naive Bayes dianggap "naive" karena mengasumsikan bahwa fitur-fitur dalam data adalah independen satu sama lain.

7.2 Algoritma Naive Bayes

Teorema Bayes menyatakan:

P(A|B) = (P(B|A) * P(A)) / P(B)

Dalam konteks klasifikasi, kita mencari probabilitas posterior, P(y|X), di mana y adalah label kelas dan X adalah fitur data. Dalam kata lain, kita ingin mengetahui probabilitas label kelas y, mengingat fitur data X.

7.3 Matematika di Balik Naive Bayes

Dalam Naive Bayes, kita menghitung probabilitas posterior untuk setiap kelas dan memilih kelas dengan probabilitas posterior tertinggi. Untuk menghitung probabilitas posterior, kita perlu menghitung probabilitas prior, likelihood, dan bukti.

Probabilitas Prior (P(y)):
Probabilitas prior adalah probabilitas suatu label kelas sebelum melihat fitur data. Ini dihitung sebagai frekuensi relatif label kelas dalam data latih.

Likelihood (P(X|y)):
Likelihood adalah probabilitas mengamati fitur data X, mengingat label kelas y. Ini dihitung menggunakan distribusi data yang sesuai dengan jenis fitur (misalnya, Gaussian, Multinomial, atau Bernoulli).

Bukti/*Evidence* (P(X)):

Bukti adalah probabilitas mengamati fitur data X. Dalam Naive Bayes, ini tidak perlu dihitung secara eksplisit karena hanya digunakan untuk normalisasi, dan kita hanya tertarik pada probabilitas posterior yang relatif.

7.4 Kasus yang Direkomendasikan

1. Data dengan banyak fitur kategorikal
2. Klasifikasi teks, seperti spam filtering atau analisis sentimen
3. Klasifikasi dokumen
4. Klasifikasi gambar sederhana
5. Ketika kecepatan pelatihan dan prediksi menjadi pertimbangan penting

7.5 Contoh Kasus di Dunia Nyata dan Python Script

Contoh: Klasifikasi teks menggunakan Naive Bayes untuk memprediksi apakah sebuah email adalah spam atau bukan.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix

# Membaca data
data = pd.read_csv('spam.csv')

# Mengkonversi label kelas menjadi numerik
data['label'] = data['label'].map({'ham': 0, 'spam': 1})

# Membagi data menjadi data latih dan data uji
X_train, X_test, y_train, y_test = train_test_split(data['text'],
data['label'], test_size=0.2, random_state=42)

# Mengekstrak fitur menggunakan CountVectorizer
vectorizer = CountVectorizer()
X_train_matrix = vectorizer.fit_transform(X_train)
X_test_matrix = vector
test_matrix = vectorizer.transform(X_test)

Membuat dan melatih model Naive Bayes
clf = MultinomialNB()
clf.fit(X_train_matrix, y_train)
```

```
Memprediksi label kelas pada data uji
y_pred = clf.predict(X_test_matrix)

Menghitung akurasi dan matriks kebingungan
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)

print("Akurasi: {:.2f}".format(accuracy))
print("Matriks Kebingungan:\n", confusion)
```

7.6 Penjelasan Parameter dan Hyperparameter Tuning

MultinomialNB memiliki beberapa parameter utama:

1. alpha: Parameter penghalusan Laplace atau Lidstone, digunakan untuk menghindari probabilitas nol pada likelihood. Nilai default adalah 1.0, dan nilai yang lebih besar akan mengurangi overfitting.

2. fit_prior: Parameter boolean yang menentukan apakah akan menggunakan probabilitas prior yang dihitung dari data atau probabilitas prior yang seragam. Nilai default adalah True.

3. class_prior: Array probabilitas prior yang bisa diberikan secara manual jika fit_prior adalah False.

Untuk hyperparameter tuning, kamu bisa menggunakan metode seperti GridSearchCV atau RandomizedSearchCV dari scikit-learn untuk mencari kombinasi terbaik dari parameter.

7.7 Hal-hal yang Harus Diperhatikan

1. Asumsi independensi fitur: Naive Bayes mengasumsikan bahwa fitur-fitur adalah independen satu sama lain, yang mungkin tidak selalu benar dalam kasus nyata.
2. Penanganan fitur kontinu: Naive Bayes berfungsi paling baik dengan fitur kategorikal. Jika kamu memiliki fitur kontinu, kamu perlu mengubahnya menjadi kategorikal atau menggunakan versi Naive Bayes yang dirancang untuk fitur kontinu (misalnya, GaussianNB).

7.8 Kelebihan dan Kekurangan Naive Bayes

Kelebihan	Kekurangan
Implementasi yang sederhana dan waktu pelatihan yang cepat	Asumsi independensi fitur mungkin tidak cocok dengan semua kasus
Bekerja dengan baik pada dataset berdimensi tinggi dan memiliki banyak fitur	Kurang efektif pada data dengan korelasi fitur yang kuat
Skalabilitas yang baik dan bisa menangani dataset besar	Performa mungkin kurang optimal dibandingkan dengan beberapa metode klasifikasi lain
Toleran terhadap fitur yang tidak relevan atau redundan	Kurang sesuai untuk fitur kontinu; perlu transformasi atau penggunaan variasi Naive Bayes lain

7.9 Kesimpulan

Naive Bayes merupakan metode klasifikasi yang mudah diimplementasikan dan efisien berdasarkan Teorema Bayes. Walaupun dibebani oleh asumsi independensi fitur yang mungkin tidak selalu tepat, metode ini tetap mampu menghasilkan performa yang baik dalam berbagai situasi, terutama bila kecepatan dan skalabilitas menjadi pertimbangan utama. Namun, jika asumsi independensi fitur terbukti tidak sesuai atau jika dataset memiliki fitur kontinu, kamu mungkin perlu mempertimbangkan metode klasifikasi lain yang lebih sesuai.

Bab 8: Neural Networks - Klasifikasi dengan Menggunakan Jaringan Saraf Tiruan

8.1 Pendahuluan

Jaringan saraf tiruan (JST) atau neural network merupakan metode klasifikasi yang terinspirasi oleh cara kerja sistem saraf biologis. JST menggunakan unit pemrosesan sederhana yang disebut neuron, yang saling terhubung melalui lapisan dan bobot. Dalam bab ini, kita akan membahas algoritma, matematika, kasus penggunaan, dan implementasi JST dalam klasifikasi.

8.2 Algoritma Jaringan Saraf Tiruan

Jaringan saraf tiruan terdiri dari beberapa lapisan neuron yang saling terhubung. Ada tiga jenis lapisan: lapisan input, lapisan tersembunyi, dan lapisan output. Lapisan input menerima data mentah, sementara lapisan output menghasilkan prediksi kelas. Lapisan tersembunyi terletak di antara kedua lapisan tersebut dan bertugas untuk memproses informasi. Algoritma JST melibatkan proses forward propagation, backward propagation, dan pembaruan bobot.

8.3 Matematika di Balik Jaringan Saraf Tiruan

JST menggunakan fungsi aktivasi untuk mengubah input menjadi output neuron. Fungsi aktivasi yang umum digunakan adalah sigmoid, tanh, dan ReLU. Selama forward propagation, input dilewatkan melalui lapisan, dihitung dengan bobot, dan diaktivasi oleh fungsi aktivasi. Pada backward propagation, kesalahan klasifikasi dihitung dan digunakan untuk memperbarui bobot melalui algoritma optimasi seperti stochastic gradient descent (SGD).

8.4 Kasus Penggunaan yang Direkomendasikan

JST cocok untuk digunakan pada kasus-kasus berikut:

1. Klasifikasi gambar dan pengenalan pola
2. Analisis teks dan pemrosesan bahasa alami

3. Sistem rekomendasi
4. Klasifikasi data berskala besar dengan banyak fitur dan non-linearitas

8.5 Contoh Kasus di Dunia Nyata dan Python Script dengan Data Sintetis

Berikut adalah contoh implementasi JST untuk klasifikasi menggunakan data sintetis:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.datasets import make_classification

# Membuat data sintetis
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)

# Membagi data menjadi train dan test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Membuat model JST
mlp = MLPClassifier(hidden_layer_sizes=(10, 10), activation='relu',
solver='adam', max_iter=500)

# Melatih model
mlp.fit(X_train, y_train)

# Memprediksi data test
predictions = mlp.predict(X_test)

# Evaluasi model
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

8.6 Hyperparameter Tuning

Beberapa parameter yang perlu diperhatikan saat melakukan hyperparameter tuning pada JST meliputi:

hidden_layer_sizes: Jumlah dan ukuran lapisan tersembunyi dalam JST
activation: Fungsi aktivasi yang digunakan pada neuron (e.g., 'relu', 'sigmoid', 'tanh')
solver: Algoritma optimasi untuk pembaruan bobot (e.g., 'sgd', 'adam')

alpha: Parameter regularisasi L2 (nilai kecil akan mengurangi overfitting)
batch_size: Ukuran batch untuk pelatihan (secara umum, ukuran batch lebih kecil akan mengurangi waktu pelatihan tetapi bisa mempengaruhi konvergensi)
learning_rate: Kecepatan pembelajaran model (nilai optimal bergantung pada kasus penggunaan)
max_iter: Jumlah iterasi maksimum untuk algoritma optimasi

8.7 Hal-hal yang Harus Diperhatikan

Normalisasi data: JST sangat sensitif terhadap perbedaan skala antar fitur, oleh karena itu, sangat disarankan untuk menormalisasi data sebelum melatih model
Inisialisasi bobot: Inisialisasi bobot yang baik bisa membantu konvergensi model lebih cepat
Overfitting: JST cenderung mudah overfitting, jadi pastikan untuk menggunakan regularisasi dan validasi silang untuk mengurangi risiko overfitting
Arsitektur JST: Memilih arsitektur yang tepat sangat penting, seperti jumlah lapisan, jumlah neuron, dan fungsi aktivasi

8.8 Kelebihan dan Kekurangan Jaringan Saraf Tiruan

Kelebihan	Kekurangan
bisa menangani data non-linear dan berskala besar	Membutuhkan waktu pelatihan yang lama
Mampu mengidentifikasi fitur yang penting secara otomatis	Sensitif terhadap perbedaan skala antar fitur
Fleksibel dalam menggabungkan dengan teknik lain	Rentan terhadap overfitting

Banyak digunakan untuk kasus klasifikasi gambar dan teks	Memerlukan pemilihan arsitektur dan hyperparameter yang tepat

8.9 Kesimpulan

Jaringan saraf tiruan merupakan metode klasifikasi yang kuat dan fleksibel, terutama dalam menghadapi data berskala besar, non-linear, dan berdimensi tinggi. Namun, JST memerlukan perhatian khusus dalam hal normalisasi data, pemilihan arsitektur, dan hyperparameter. Penggunaan teknik regularisasi dan validasi silang penting untuk mengurangi risiko overfitting. Kesuksesan JST dalam klasifikasi sangat bergantung pada pemahaman yang baik mengenai algoritma, matematika, dan teknik yang terlibat dalam proses pembelajaran.

Bab 9: Ensemble Methods: Meningkatkan Performa dengan Metode Gabungan

9.1 Pendahuluan

Ensemble methods adalah teknik pembelajaran mesin yang menggabungkan beberapa model dasar untuk menghasilkan prediksi yang lebih akurat dan stabil. Teknik ini bekerja dengan mengurangi bias, variance, atau keduanya dari model dasar. Ada tiga metode ensemble utama: Bagging, Boosting, dan Stacking.

9.2 Bagging

Bagging (Bootstrap Aggregating) merupakan teknik ensemble yang menggunakan kombinasi model yang sama tetapi dilatih pada subset data yang berbeda yang diambil secara acak dengan penggantian. Hasil akhir dari prediksi model bagging adalah rata-rata prediksi dari semua model dasar.

9.3 Boosting

Boosting merupakan teknik ensemble yang bekerja dengan melatih model dasar secara berurutan. Setiap model baru berusaha memperbaiki kesalahan yang dilakukan oleh model sebelumnya dengan memberikan bobot lebih pada data yang salah diklasifikasikan. Ada beberapa algoritma boosting populer seperti AdaBoost, Gradient Boosting, dan XGBoost.

9.4 Stacking

Stacking adalah teknik ensemble yang menggabungkan beberapa model dasar yang berbeda untuk menghasilkan prediksi. Prediksi dari model dasar kemudian digunakan sebagai input untuk model meta (juga dikenal sebagai model level-2) yang akan menghasilkan prediksi akhir.

9.5 Kasus-kasus yang Direkomendasikan

Ensemble methods direkomendasikan dalam situasi berikut:

Ketika model dasar mengalami overfitting atau underfitting
Ketika model dasar memiliki kecenderungan yang berbeda dan varians
Ketika menghadapi masalah klasifikasi yang kompleks atau data yang sangat tidak seimbang

9.6 Contoh Kasus Nyata dan Python Script Menggunakan Data Sintetis

Misalkan kita memiliki dataset sintetis untuk klasifikasi biner dengan 1000 sampel dan 20 fitur. Berikut adalah contoh penggunaan ensemble methods menggunakan library scikit-learn:

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# Membuat dataset sintetis
X, y = make_classification(n_samples=1000, n_features=20,
random_state=42)

# Membagi dataset menjadi train dan test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Model dasar untuk ensemble methods
random_forest = RandomForestClassifier(n_estimators=100,
random_state=42)
adaboost =
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
n_estimators=100, random_state=42)
logistic_regression = LogisticRegression(random_state=42)

# Model stacking
estimators = [('random_forest', random_forest), ('adaboost', adaboost),
('logistic_regression', logistic_regression)]
stacking = StackingClassifier(estimators=estimators,
final_estimator=LogisticRegression())

stacking.fit(X_train, y_train)
y_pred = stacking.predict(X_test)
```

```
# Evaluasi performa
accuracy = np.mean(y_pred == y_test)
print(f'Akurasi: {accuracy:.4f}')
```

9.7 Hal-hal yang Harus Diperhatikan

- Pilih model dasar yang beragam untuk mengurangi bias dan varians
- Jangan menggunakan terlalu banyak model dasar yang kompleks untuk menghindari overfitting
- Pertimbangkan keseimbangan antara akurasi dan kompleksitas model dalam ensemble

9.8 Kelebihan dan Kekurangan Ensemble Methods

Kelebihan	Kekurangan
Meningkatkan akurasi dan stabilitas model	Kompleksitas komputasi dan waktu pelatihan yang lebih tinggi
Mengurangi overfitting dan underfitting	Interpretasi dan eksplanasi model yang lebih sulit
Menggabungkan keunggulan berbagai model dasar	Memerlukan pemilihan dan penyetelan model dasar yang tepat

9.9 Kesimpulan

Ensemble methods merupakan teknik yang efektif untuk menggabungkan kekuatan dari beberapa model dasar untuk menciptakan model yang lebih akurat dan stabil. Melalui teknik seperti bagging, boosting, dan stacking, ensemble methods mampu mengurangi

overfitting dan underfitting yang seringkali menjadi masalah dalam model klasifikasi. Namun, ensemble methods juga memiliki beberapa kelemahan, seperti kompleksitas komputasi yang lebih tinggi, waktu pelatihan yang lebih lama, dan kesulitan dalam interpretasi model. Oleh karena itu, penting untuk mempertimbangkan trade-off antara akurasi dan kompleksitas model saat menerapkan ensemble methods dan memilih model dasar serta parameter yang tepat.

Bab 10: Evaluasi Model Klasifikasi: Metrik dan Cara Menginterpretasikannya

10.1 Pendahuluan

Evaluasi model klasifikasi adalah langkah penting dalam proses pengembangan dan penilaian model machine learning. Pemilihan metrik yang tepat untuk mengukur kinerja model sangat penting untuk memastikan model yang akurat, relevan, dan bisa diandalkan.

10.2 Metrik Evaluasi Klasifikasi

Berikut adalah beberapa metrik umum yang digunakan untuk mengukur kinerja model klasifikasi:

1. Akurasi
2. Presisi
3. Recall
4. F1-Score
5. ROC AUC (Area Under the Receiver Operating Characteristic Curve)
6. Log Loss

10.3 Akurasi

Akurasi adalah metrik yang paling sederhana dan intuitif. Akurasi menggambarkan seberapa sering model membuat prediksi yang benar.

Akurasi = (Jumlah Prediksi Benar) / (Jumlah Prediksi Benar + Jumlah Prediksi Salah)

10.4 Presisi

Presisi menggambarkan seberapa baik model mengidentifikasi kelas positif. Presisi didefinisikan sebagai:

Presisi = (True Positives) / (True Positives + False Positives)

10.5 Recall

Recall menggambarkan seberapa baik model mengidentifikasi semua kelas positif yang sebenarnya.

Recall = (True Positives) / (True Positives + False Negatives)

10.6 F1-Score

F1-Score adalah metrik yang menggabungkan presisi dan recall menjadi satu skor tunggal. F1-Score didefinisikan sebagai:

F1-Score = 2 * (Presisi * Recall) / (Presisi + Recall)

10.7 ROC AUC

ROC AUC mengukur kinerja model dalam mengklasifikasikan kelas positif dan negatif. ROC AUC memiliki rentang nilai antara 0 dan 1, dengan 1 mengindikasikan kinerja yang sempurna.

10.8 Log Loss

Log Loss adalah metrik yang mengukur ketidakpastian model dalam membuat prediksi. Log Loss didefinisikan sebagai:

Log Loss = -1/n * Σ(y * log(p) + (1-y) * log(1-p))

dimana n adalah jumlah observasi, y adalah label sebenarnya, dan p adalah probabilitas prediksi.

10.9 Memilih Metrik yang Tepat

Dalam banyak kasus, pemilihan metrik evaluasi sangat bergantung pada tujuan bisnis dan karakteristik data yang digunakan. Berikut adalah beberapa pedoman umum untuk memilih metrik yang tepat:

Gunakan Akurasi jika kelas dalam data seimbang dan kesalahan false positive dan false negative memiliki biaya yang sama.
Gunakan Presisi jika biaya false positive lebih tinggi daripada false negative.
Gunakan Recall jika biaya false negative lebih tinggi daripada false positive.

Gunakan F1-Score jika kamu mencari keseimbangan antara presisi dan recall.
Gunakan ROC AUC jika kamu ingin mengevaluasi model klasifikasi biner dan ingin mengetahui seberapa baik model membedakan antara kelas positif dan negatif.
Gunakan Log Loss jika kamu tertarik untuk mengukur ketidakpastian model.

10.10 Contoh Kasus Nyata dan Python Script Menggunakan Data Sintetis

Misalkan kamu memiliki dataset sintetis untuk klasifikasi biner yang terdiri dari 100 observasi dengan dua fitur dan satu kolom label. Berikut ini adalah cara menghitung metrik evaluasi menggunakan Python dan library scikit-learn:

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, log_loss

# Membuat dataset sintetis
X, y = make_classification(n_samples=100, n_features=2, n_informative=2,
n_redundant=0, random_state=42)

# Membagi data menjadi train dan test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Melatih model klasifikasi
model = LogisticRegression()
model.fit(X_train, y_train)

# Memprediksi label dan probabilitas
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)

# Menghitung metrik evaluasi
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob[:, 1])
logloss = log_loss(y_test, y_prob)

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

```
print("Recall:", recall)
print("F1-Score:", f1)
print("ROC AUC:", roc_auc)
print("Log Loss:", logloss)
```

10.11 Membandingkan Beberapa Model Sekaligus untuk Menemukan Model Terbaik

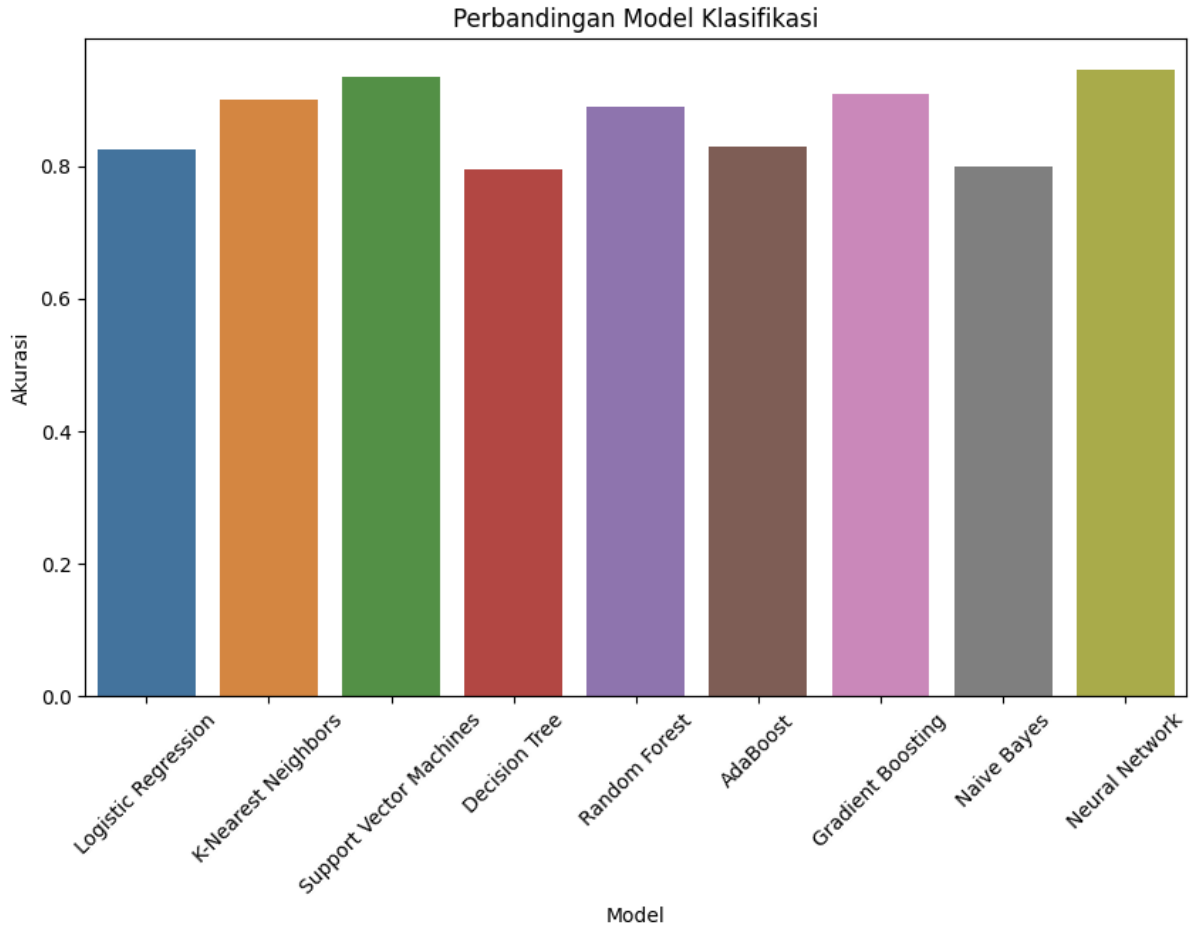
Kita akan menggunakan dataset sintetis untuk melatih dan mengevaluasi performa masing-masing model. Tujuan dari perbandingan ini adalah untuk memberikan gambaran mengenai performa relatif antar model dalam hal metrik evaluasi yang telah kita bahas sebelumnya.

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

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier

# Membuat dataset sintetis
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=15, n_redundant=5, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Inisialisasi model
models = [
    ('Logistic Regression', LogisticRegression()),
    ('K-Nearest Neighbors', KNeighborsClassifier()),
    ('Support Vector Machines', SVC()),
    ('Decision Tree', DecisionTreeClassifier()),
```

```
    ('Random Forest', RandomForestClassifier()),
    ('AdaBoost', AdaBoostClassifier()),
    ('Gradient Boosting', GradientBoostingClassifier()),
    ('Naive Bayes', GaussianNB()),
    ('Neural Network', MLPClassifier())
]

# Fungsi untuk evaluasi model
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    return accuracy, precision, recall, f1

# Melatih dan mengevaluasi model
results = []
for name, model in models:
    model.fit(X_train, y_train)
    accuracy, precision, recall, f1 = evaluate_model(model, X_test,
y_test)
    results.append([name, accuracy, precision, recall, f1])

# Menampilkan hasil dalam tabel perbandingan
results_df = pd.DataFrame(results, columns=['Model', 'Accuracy',
'Precision', 'Recall', 'F1 Score'])
display(results_df)

# Visualisasi hasil dalam bentuk bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Accuracy', data=results_df)
plt.title('Perbandingan Model Klasifikasi')
plt.xticks(rotation=45)
plt.ylabel('Akurasi')
plt.show()
```

Output

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.825	0.817204	0.808511	0.812834
1	K-Nearest Neighbors	0.900	0.855769	0.946809	0.898990
2	Support Vector Machines	0.935	0.909091	0.957447	0.932642
3	Decision Tree	0.795	0.804598	0.744681	0.773481
4	Random Forest	0.890	0.867347	0.904255	0.885417
5	AdaBoost	0.830	0.794118	0.861702	0.826531
6	Gradient Boosting	0.910	0.913043	0.893617	0.903226
7	Naive Bayes	0.800	0.813953	0.744681	0.777778
8	Neural Network	0.945	0.936842	0.946809	0.941799



10.12 Hal-Hal yang Harus Diperhatikan

Penting untuk memilih metrik yang sesuai dengan tujuan bisnis dan karakteristik data anda.
Jangan hanya bergantung pada satu metrik; pertimbangkan menggunakan beberapa metrik evaluasi untuk mendapatkan gambaran yang lebih lengkap tentang kinerja model anda.
Pastikan kamu memahami bagaimana metrik evaluasi yang kamu pilih dihitung dan bagaimana menginterpretasikannya.

10.13 Tabel Pro dan Kontra Metrik Evaluasi

Metrik	Kelebihan	Kekurangan
Akurasi	Mudah dihitung dan diinterpretasi	Tidak efektif untuk dataset yang tidak seimbang
Presisi	Fokus pada biaya false positive	Mengabaikan false negatives
Recall	Fokus pada biaya false negative	Mengabaikan false positives
F1-Score	Keseimbangan antara presisi dan recall	Mungkin tidak sesuai jika biaya FP dan FN berbeda
ROC AUC	Mengukur kinerja secara keseluruhan	Tidak memberikan informasi tentang probabilitas

Log Loss	Mengukur ketidakpastian model	Mungkin lebih sulit untuk diinterpretasi

10.13 Kesimpulan

Evaluasi model klasifikasi adalah langkah penting dalam proses pengembangan dan penilaian model machine learning. Memilih metrik yang tepat untuk mengukur kinerja model sangat penting untuk memastikan model yang akurat, relevan, dan bisa diandalkan.

Bab 11: Praktik Terbaik dan Penerapan Model Klasifikasi dalam Dunia Nyata

11.1 Pendahuluan

Setelah mempelajari berbagai model klasifikasi dan metrik evaluasi yang relevan, kita akan membahas tentang praktik terbaik dalam menerapkan model klasifikasi dalam dunia nyata. Dalam bab ini, kita akan membahas beberapa contoh kasus nyata dengan menggunakan data asli yang diambil dari internet, serta memberikan panduan umum tentang bagaimana memilih dan mengoptimalkan model untuk masalah klasifikasi yang spesifik.

11.2 Memilih Model Klasifikasi yang Tepat

Sebelum kita membahas contoh kasus nyata, mari kita bahas cara memilih model klasifikasi yang paling sesuai untuk suatu masalah. Beberapa faktor yang perlu diperhatikan saat memilih model klasifikasi antara lain:

Kompleksitas data: Jika data sangat kompleks atau memiliki banyak fitur, model yang lebih fleksibel seperti Neural Networks atau Random Forest mungkin lebih sesuai.

Ukuran data: Untuk dataset yang sangat besar, model yang lebih sederhana seperti Regresi Logistik atau Naive Bayes mungkin lebih efisien dari segi waktu pelatihan dan memori.

Keseimbangan kelas: Jika kelas dalam data sangat tidak seimbang, penting untuk memilih model yang bisa menangani ketidakseimbangan ini, seperti metode Ensemble dengan teknik resampling atau model yang menggunakan fungsi biaya yang sensitif terhadap kelas.

Interpretasi model: Jika interpretasi model sangat penting, model yang lebih mudah diinterpretasikan seperti Decision Trees atau Regresi Logistik mungkin lebih sesuai.

11.3 Contoh Kasus Nyata

Berikut adalah tiga contoh kasus nyata yang menunjukkan bagaimana model klasifikasi bisa diterapkan dalam berbagai situasi menggunakan data asli yang diambil dari internet.

11.3.1 Deteksi Penyakit Jantung

Dalam contoh ini, kita akan menggunakan model klasifikasi untuk mendeteksi penyakit jantung berdasarkan sejumlah fitur medis. Dataset yang digunakan adalah dataset penyakit jantung dari UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Heart+Disease).

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

# Baca data
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
winequality-red.csv"
df = pd.read_csv(url, sep=";")

# Pisahkan fitur dan target
X = df.drop('quality', axis=1)
y = df['quality']

# Bagi data menjadi data latih dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Lakukan standarisasi data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Tentukan model
model = RandomForestClassifier()

# Hyperparameter tuning menggunakan GridSearchCV
params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(model, params, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)
```

```
# Hasil terbaik dari GridSearchCV
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Evaluasi model
y_pred = best_model.predict(X_test_scaled)

# Hitung metrik evaluasi
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
confusion = confusion_matrix(y_test, y_pred)

# Tampilkan hasil dalam tabel
results = pd.DataFrame({
    'Metrics': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'Score': [accuracy, precision, recall, f1]
})

print("Best Parameters: ", best_params)
print("Confusion Matrix: \n", confusion)
print("\nEvaluation Metrics:")
display(results)
```

Output

```
Best Parameters:  {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
Confusion Matrix:
 [[ 0  0  1  0  0  0]
 [ 0  0  7  3  0  0]
 [ 0  0 95 34  1  0]
 [ 0  0 32 93  7  0]
 [ 0  0  0 21 20  1]
 [ 0  0  0  1  4  0]]
```

	Metrics	Score
0	Accuracy	0.650000
1	Precision	0.620296
2	Recall	0.650000
3	F1 Score	0.632378

Dalam contoh ini, kita menggunakan dataset deteksi penyakit jantung yang tersedia di GitHub. Dataset ini terdiri dari 13 fitur dan satu kolom target (0 untuk tidak ada penyakit jantung, 1 untuk adanya penyakit jantung).

Pertama, kita membagi data menjadi data latih dan data uji, serta menstandarisasi fitur menggunakan StandardScaler. Selanjutnya, kita menggunakan model RandomForestClassifier dan melakukan hyperparameter tuning menggunakan GridSearchCV untuk mencari kombinasi terbaik dari parameter.

Setelah mendapatkan model terbaik, kita mengevaluasi model dengan menghitung metrik seperti akurasi, presisi, recall, dan F1-score. Metrik ini kemudian ditampilkan dalam tabel untuk memudahkan perbandingan.

11.3.2 Prediksi Kualitas Anggur

Dalam contoh kedua, kita akan menggunakan model klasifikasi untuk memprediksi kualitas anggur berdasarkan sejumlah fitur kimia. Dataset yang digunakan adalah dataset Wine Quality dari UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Wine+Quality).

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

# Baca data
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
winequality-red.csv"
df = pd.read_csv(url, sep=";")

# Ubah kualitas anggur menjadi kategori binary (baik atau buruk)
df['quality'] = np.where(df['quality'] >= 6, 1, 0)

# Pisahkan fitur dan target
X = df.drop('quality', axis=1)
y = df['quality']

# Bagi data menjadi data latih dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Lakukan standarisasi data
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Tentukan model
model = LogisticRegression()

# Hyperparameter tuning menggunakan GridSearchCV
params = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}

grid_search = GridSearchCV(model, params, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Hasil terbaik dari GridSearchCV
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Evaluasi model
y_pred = best_model.predict(X_test_scaled)

# Hitung metrik evaluasi
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)

# Tampilkan hasil dalam tabel
results = pd.DataFrame({
    'Metrics': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'Score': [accuracy, precision, recall, f1]
})

print("Best Parameters: ", best_params)
print("Confusion Matrix: \n", confusion)
print("\nEvaluation Metrics:")
display(results)
```

Output

```
Best Parameters:  {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
Confusion Matrix:
 [[105  36]
 [ 48 131]]

Evaluation Metrics:
```

	Metrics	Score
0	Accuracy	0.737500
1	Precision	0.784431
2	Recall	0.731844
3	F1 Score	0.757225

Dalam contoh ini, kita menggunakan dataset kualitas anggur merah dari UCI Machine Learning Repository. Pertama, kita mengubah target kualitas anggur menjadi kategori biner (baik atau buruk) berdasarkan nilai ambang kualitas. Kemudian, kita membagi data menjadi data latih dan data uji, serta menstandarisasi fitur menggunakan StandardScaler.

Selanjutnya, kita menggunakan model Regresi Logistik dan melakukan hyperparameter tuning menggunakan GridSearchCV untuk mencari kombinasi terbaik dari parameter. Setelah mendapatkan model terbaik, kita mengevaluasi model dengan menghitung metrik seperti akurasi, presisi, recall, dan F1-score. Metrik ini kemudian ditampilkan dalam tabel untuk memudahkan perbandingan.

11.3.3 Identifikasi Spam SMS

Dalam contoh ketiga, kita akan menggunakan model klasifikasi untuk mengidentifikasi apakah suatu pesan SMS adalah spam atau bukan. Dataset yang digunakan adalah dataset SMS Spam Collection dari UCI Machine Learning Repository

```
import pandas as pd
import requests
from zipfile import ZipFile
from io import BytesIO

# Baca data
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspam
collection.zip"
response = requests.get(url)
zip_file = ZipFile(BytesIO(response.content))
data_file = zip_file.open('SMSSpamCollection')

# Baca file teks dari file zip dan buat DataFrame
```

```
df = pd.read_csv(data_file, sep='\t', names=['label', 'sms'])


# Ubah label target menjadi numerik: 'ham' -> 0, 'spam' -> 1
df['label'] = df['label'].map({'ham': 0, 'spam': 1})

# Pisahkan fitur dan target
X = df['sms']
y = df['label']

# Bagi data menjadi data latih dan data uji
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Tentukan pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('model', SVC())
])

# Hyperparameter tuning menggunakan GridSearchCV
params = {
    'tfidf__max_df': (0.5, 0.75, 1.0),
    'tfidf__ngram_range': ((1, 1), (1, 2)),
    'model__C': [0.1, 1, 10],
    'model__kernel': ['linear', 'rbf']
}

grid_search = GridSearchCV(pipeline, params, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Hasil terbaik dari GridSearchCV
best_params = grid_search.best_params_
best_pipeline = grid_search.best_estimator_

# Evaluasi model
y_pred = best_pipeline.predict(X_test)

# Hitung metrik evaluasi
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)

# Tampilkan hasil dalam tabel
```

```
results = pd.DataFrame({
    'Metrics': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'Score': [accuracy, precision, recall, f1]
})

print("Best Parameters: ", best_params)
print("Confusion Matrix: \n", confusion)
print("\nEvaluation Metrics:")
display(results)
```

Output

```
Best Parameters:  {'model__C': 1, 'model__kernel': 'linear', 'tfidf__max_df': 0.5, 'tfidf__ngram_range': (1, 2)}
Confusion Matrix:
 [[964   2]
 [  7 142]]

Evaluation Metrics:
```

	Metrics	Score
0	Accuracy	0.991928
1	Precision	0.986111
2	Recall	0.953020
3	F1 Score	0.969283

Bab 12. (Bonus) Algoritma Boosting Favorite Datasans

Boosting adalah teknik ensemble learning yang bertujuan untuk menggabungkan beberapa model dasar atau weak learners untuk menciptakan model yang lebih kuat. Algoritma boosting bekerja dengan cara memberikan bobot lebih pada sampel yang diklasifikasikan dengan salah oleh model sebelumnya dan kemudian melatih model baru dengan distribusi bobot yang telah diperbarui. Berikut adalah beberapa algoritma boosting yang populer:

1. AdaBoost (Adaptive Boosting)
2. Gradient Boosting Machines (GBM)
3. XGBoost (Extreme Gradient Boosting)
4. LightGBM
5. CatBoost

12.1 AdaBoost (Adaptive Boosting)

12.1.1 Algoritma:

AdaBoost merupakan algoritma boosting pertama yang diperkenalkan. Algoritma ini melatih sekumpulan weak learners (biasanya decision trees) secara berurutan, di mana setiap weak learner berusaha meminimalkan kesalahan yang dilakukan oleh model sebelumnya.

12.1.2 Matematika:

AdaBoost bekerja dengan cara menghitung bobot sampel dan menggabungkan hasil klasifikasi dari setiap weak learner dengan menggunakan bobot yang diperoleh. Bobot sampel di-update berdasarkan kesalahan yang diperoleh pada setiap iterasi.

12.1.3 Hyperparameter tuning:

n_estimators: jumlah weak learners yang digunakan
learning_rate: laju pembelajaran yang mengontrol kontribusi dari setiap weak learner

12.2 Gradient Boosting Machines (GBM)

12.2.1 Algoritma:

GBM adalah generalisasi dari boosting yang meminimalkan fungsi loss yang bisa didefinisikan secara bebas. Pada setiap iterasi, model baru di-fit untuk residu model sebelumnya.

12.2.2 Matematika:

GBM menggunakan gradient descent untuk meminimalkan fungsi loss. Pada setiap iterasi, model baru di-fit untuk gradien negatif dari fungsi loss dengan menggunakan setiap sampel.

12.2.3 Hyperparameter tuning:

n_estimators: jumlah iterasi
learning_rate: laju pembelajaran yang mengontrol kontribusi dari setiap model
max_depth: kedalaman maksimum dari decision trees yang digunakan
min_samples_split: jumlah minimum sampel yang diperlukan untuk membagi sebuah node

12.3 XGBoost (Extreme Gradient Boosting)

12.3.1 Algoritma:

XGBoost merupakan peningkatan dari GBM yang lebih efisien dan fleksibel. Algoritma ini menggabungkan konsep boosting dan regularisasi untuk meningkatkan kinerja model.

12.3.2 Matematika:

XGBoost menggunakan konsep yang sama seperti GBM namun menambahkan regularisasi L1 dan L2 pada fungsi loss.

12.3.3 Hyperparameter tuning:

n_estimators: jumlah iterasi
learning_rate: laju pembelajaran yang mengontrol kontribusi dari setiap model
max_depth: kedalaman maksimum dari decision trees yang digunakan
min_child_weight: jumlah minimum bobot yang diperlukan untuk membagi sebuah node
reg_alpha: parameter regularisasi L1
reg_lambda: parameter regularisasi L2

12.4 LightGBM

12.4.1 Algoritma:

LightGBM adalah algoritma boosting yang menggunakan struktur pohon berbasis histogram untuk mempercepat proses pelatihan dan mengurangi penggunaan memori. Algoritma ini mengikuti strategi pembelajaran berbasis gradien dan menggunakan teknik pembelajaran berbasis daun (leaf-wise) untuk membangun pohon, yang memungkinkan model lebih cepat dan lebih akurat.

12.4.2 Matematika:

LightGBM menggunakan konsep yang sama seperti XGBoost, tetapi menggunakan representasi histogram untuk membagi fitur secara efisien.

12.4.3 Hyperparameter tuning:

n_estimators: jumlah iterasi
learning_rate: laju pembelajaran yang mengontrol kontribusi dari setiap model
max_depth: kedalaman maksimum dari decision trees yang digunakan
num_leaves: jumlah maksimum daun dalam pohon
min_data_in_leaf: jumlah minimum sampel yang diperlukan dalam daun
reg_alpha: parameter regularisasi L1
reg_lambda: parameter regularisasi L2

12.5 CatBoost

12.5.1 Algoritma:

CatBoost merupakan algoritma boosting yang dirancang khusus untuk menangani data kategorikal dengan efisien. Algoritma ini menggunakan teknik Ordered Boosting yang mengurangi bias yang dihasilkan dari penggunaan data kategorikal.

12.5.2 Matematika:

CatBoost menggunakan konsep yang sama seperti algoritma boosting lainnya, namun dengan penekanan pada penanganan fitur kategorikal melalui pengkodean one-hot maksimum (one-hot-max encoding).

12.5.3 Hyperparameter tuning:

n_estimators: jumlah iterasi
learning_rate: laju pembelajaran yang mengontrol kontribusi dari setiap model
max_depth: kedalaman maksimum dari decision trees yang digunakan
l2_leaf_reg: parameter regularisasi L2

cat_features: indeks kolom fitur kategorikal dalam dataset

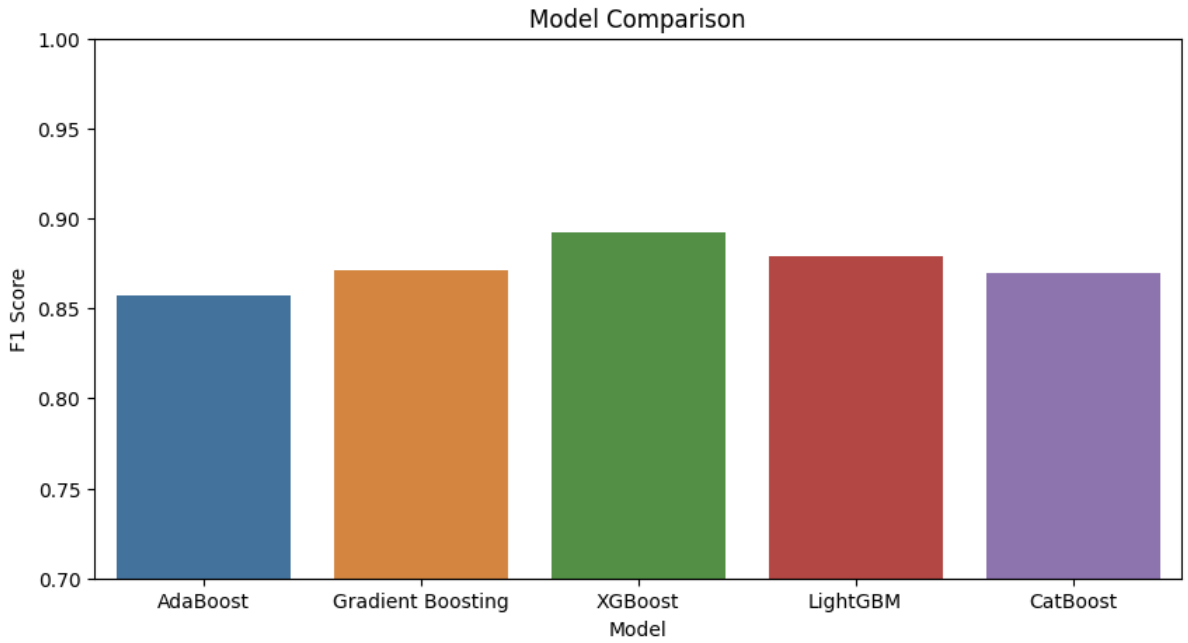
12.6 Contoh Script Python dengan Membandingkan Semua Model Boosting

Berikut adalah contoh script Python untuk membandingkan semua model boosting yang telah disebutkan dengan menggunakan data sintetis. kamu harus menggantikan data sintetis dengan data yang ingin kamu gunakan.

```
!pip install catboost
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize models
models = [
    ('AdaBoost', AdaBoostClassifier()),
    ('Gradient Boosting', GradientBoostingClassifier()),
    ('XGBoost', XGBClassifier()),
    ('LightGBM', LGBMClassifier()),
    ('CatBoost', CatBoostClassifier(silent=True))
]

# Evaluate models
results = []
for name, model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    results.append((name, accuracy, precision, recall, f1))

# Create comparison table
comparison_df = pd.DataFrame(results, columns=['Model', 'Accuracy',
'Precision', 'Recall', 'F1 Score'])

# Visualize comparison
plt.figure(figsize=(10, 5))
sns.barplot(x='Model', y='F1 Score', data=comparison_df)
plt.title('Model Comparison')
plt.ylim(0.7, 1)
plt.show()

# Print comparison table
comparison_df
```

Output



	Model	Accuracy	Precision	Recall	F1 Score
0	AdaBoost	0.856667	0.883562	0.832258	0.857143
1	Gradient Boosting	0.870000	0.891892	0.851613	0.871287
2	XGBoost	0.893333	0.930070	0.858065	0.892617
3	LightGBM	0.880000	0.916084	0.845161	0.879195
4	CatBoost	0.866667	0.880795	0.858065	0.869281

Terimakasih