# LAPORAN PRAKTIKUM
# PRAKTIKUM PENAMBANGAN DATA
# PERTEMUAN 3
# CLASSIFICATION


Disusun oleh:

| | |
|---|---|
| Nama | : Najmina Kinanti Putri |
| NIM | : 23/514821/SV/22432 |
| Kelas | : PL5A2 |
| Dosen Pengampu | : Dr.Eng. Ganjar Alfian, S.T., M.Eng. |

PROGRAM STUDI D-IV TEKNOLOGI REKAYASA PERANGKAT LUNAK
DEPARTEMEN TEKNIK ELEKTRO DAN INFORMATIKA
SEKOLAH VOKASI
UNIVERSITAS GADJAH MADA
YOGYAKARTA

2025

# PERTEMUAN 3
# CLASSIFICATION

## A. Tujuan Percobaan

1. Mahasiswa dapat memahami konsep klasifikasi dalam *machine learning*
2. Mahasiswa dapat memahami dan membandingkan Logistic Regression, KNN, Decision Tree, Random Forest, dan AdaBoost.
3. Mahasiswa dapat menganalisis kelebihan, kekurangan, serta performa tiap model

## B. Dasar Teori

### 1. Classification

Klasifikasi merupakan salah satu metode penting dalam pembelajaran mesin yang bertujuan untuk mengelompokkan data ke dalam kelas tertentu berdasarkan karakteristiknya. Dalam ranah *text classification*, tugas ini didefinisikan sebagai proses "selecting labels that are relevant to the content of a document" [1]. Dengan kata lain, klasifikasi berfungsi untuk memberikan struktur eksplisit pada data tidak terstruktur sehingga dapat dimanfaatkan lebih lanjut dalam berbagai analisis, seperti pengelompokan topik, pemetaan minat pengguna, maupun analisis sentimen.

### 2. Logistic Regression

Logistic Regression merupakan salah satu algoritma pembelajaran mesin yang populer dan banyak digunakan dalam tugas klasifikasi, terutama untuk masalah biner. Model ini bekerja dengan memodelkan hubungan antara fitur input dan label kelas menggunakan fungsi logistik (sigmoid), sehingga hasil prediksi dapat diinterpretasikan sebagai probabilitas. Dalam konteks klasifikasi dokumen, Logistic Regression digunakan untuk memetakan fitur-fitur yang diekstraksi dari dokumen ke dalam kelas tertentu. Seperti yang dijelaskan oleh Nair *et al*. bahwa "logistic regression can be used to model the relationship between input features extracted from the document images and the corresponding class labels" [2]. Dengan pendekatan ini, Logistic Regression mampu memberikan hasil klasifikasi yang akurat dan lebih mudah diinterpretasikan dibandingkan dengan metode regresi linear ganda.

### 3. K Nearest Neighbour

K-Nearest Neighbor (KNN) merupakan salah satu algoritma klasifikasi yang sederhana namun banyak digunakan dalam pembelajaran mesin. Prinsip dasar algoritma ini adalah menentukan kelas suatu data baru berdasarkan kemiripannya dengan sejumlah data tetangga terdekat dalam dataset. Algoritma ini bekerja dengan menghitung jarak antara data baru dengan data dalam dataset, kemudian memilih k tetangga terdekat untuk menentukan kelas mayoritas. Pemilihan nilai k sangat berpengaruh terhadap performa, karena nilai k yang terlalu kecil dapat membuat model sensitif terhadap noise, sedangkan nilai k yang terlalu besar dapat menurunkan akurasi. Berbagai modifikasi juga telah dikembangkan, salah satunya *Weighted* KNN yang memberikan bobot lebih besar pada tetangga yang lebih dekat agar hasil klasifikasi lebih akurat [3].

### 4. Decision Tree

Decision Tree merupakan salah satu algoritma pembelajaran mesin yang digunakan untuk klasifikasi maupun regresi. Algoritma ini bekerja dengan membagi data ke dalam cabang-cabang berdasarkan atribut tertentu hingga mencapai keputusan pada simpul daun (leaf node). Setiap simpul internal merepresentasikan suatu tes terhadap atribut, cabang menunjukkan hasil dari tes, dan simpul daun berisi label kelas. Proses pembangunan Decision Tree biasanya menggunakan kriteria tertentu seperti Information Gain, Gini Index, atau Chi-Square untuk menentukan atribut yang paling baik dalam memisahkan data. Keunggulan algoritma ini terletak pada kemudahan interpretasi, karena struktur pohon yang dihasilkan menyerupai proses pengambilan keputusan manusia. Namun, Decision Tree juga memiliki kelemahan yaitu rentan mengalami overfitting apabila pohon yang dibentuk terlalu dalam, sehingga diperlukan teknik seperti pruning untuk meningkatkan generalisasi model [4].

### 5. Random Forest

Random Forest adalah algoritma *ensemble learning* berbasis pohon keputusan yang menggabungkan banyak pohon untuk meningkatkan akurasi prediksi dan mengurangi risiko *overfitting*. Setiap pohon dibangun menggunakan sampel acak dari data pelatihan dan subset acak dari fitur, menghasilkan model yang beragam. Prediksi akhir diperoleh melalui agregasi

hasil dari semua pohon, biasanya dengan mayoritas suara untuk klasifikasi atau rata-rata untuk regresi. Schonlau dan Zou menjelaskan bahwa algoritma ini secara efektif meningkatkan prediksi dengan membangun sejumlah pohon keputusan secara paralel dan menggabungkan hasilnya untuk mendapatkan prediksi yang lebih stabil dan akurat [5].

Keunggulan utama Random Forest meliputi kemampuannya dalam menangani data berdimensi tinggi, ketahanannya terhadap *overfitting*, serta kemampuannya dalam mengukur pentingnya variabel (*feature importance*). Algoritma ini juga efektif dalam menangani data yang tidak seimbang dan dapat digunakan untuk berbagai tugas seperti klasifikasi, regresi, dan deteksi anomali. Yates dan Islam menyarankan bahwa Random Forest sangat cocok untuk meningkatkan kecepatan pemrosesan tanpa mengorbankan akurasi, yang menjadikannya pilihan yang efisien dalam banyak aplikasi analisis data [6].

**6. AdaBoost**

AdaBoost (*Adaptive Boosting*) adalah salah satu algoritma *ensemble learning* yang bekerja dengan menggabungkan sejumlah *weak classifiers* untuk membentuk model prediksi yang lebih kuat. Pada setiap iterasi, algoritma ini menyesuaikan bobot dari data pelatihan, sehingga sampel yang salah klasifikasi pada iterasi sebelumnya akan mendapat bobot lebih tinggi pada iterasi berikutnya. Dengan cara ini, model berfokus pada data yang sulit diklasifikasikan. Keunggulan utama AdaBoost adalah kemampuannya meningkatkan akurasi klasifikasi dengan meminimalisir kesalahan dari *weak classifiers*. Algoritma ini banyak digunakan dalam berbagai bidang, termasuk klasifikasi medis, pengenalan wajah, dan deteksi objek. Namun, kelemahan dari AdaBoost adalah sensitif terhadap *outliers* dan data dengan noise tinggi, karena bobot pada data salah klasifikasi dapat meningkat drastis [7].

**7. Evaluation Metric**

*Evaluation Metric* digunakan untuk mengukur dan merangkum performa model klasifikasi ketika diuji terhadap data uji. Salah satu pendekatan evaluasi yang paling umum adalah melalui *confusion matrix*, yang menyajikan hubungan antara hasil prediksi dan nilai aktual. Dalam matriks ini, baris merepresentasikan hasil prediksi model, sedangkan kolom menunjukkan nilai aktual dari data. Nilai aktual dikodekan dengan (1) untuk kelas positif

dan (0) untuk kelas negatif. Sementara itu, hasil prediksi ditandai dengan (1) jika model memprediksi positif, dan (0) jika memprediksi negatif [8].

Komponen utama dari confusion matrix mencakup *True Positive* (TP) menunjukkan jumlah data positif yang berhasil diklasifikasikan dengan benar, *True Negative* (TN) menunjukkan jumlah data negatif yang diprediksi dengan tepat, *False Positive* (FP) menunjukkan data negatif yang secara keliru diklasifikasikan sebagai positif (Type I error), dan *False Negative* (FN) menunjukkan data positif yang secara salah diklasifikasikan sebagai negatif (Type II error) [16].

| | | Predicted Values | |
|---|---|---|---|
| | | Positive | Negative |
| Actual Values | Positive | TP | FN |
| | Negative | FP | TN |

Metrik ini dijadikan dasar dalam menghitung ukuran kinerja lainnya seperti *accuracy*, *precision*, *recall*, dan *F1-score*. Berikut adalah rumus untuk menghitung *confusion matrix*. Nilai *accuracy* diperoleh menggunakan rumus 1. Nilai *precision* diperoleh menggunakan rumus 2. Nilai *recall* diperoleh menggunakan rumus 3. Nilai *F1-score* diperoleh menggunakan rumus 4.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F1\text{-}score = 2 \times \frac{Precission \times Recall}{Precission + Recall} \quad (4)$$

*Accuracy* mengukur proporsi prediksi yang benar dari keseluruhan prediksi. *Precision* mengukur seberapa banyak dari seluruh prediksi positif yang benar. *Recall* menunjukkan seberapa banyak data positif yang berhasil dikenali oleh model. *F1-score* adalah rata-rata harmonis dari *precision* dan *recall*, yang digunakan untuk menyeimbangkan keduanya [8].

## C. Hasil dan Pembahasan

1. **Tugas 1**

- ***Download* dataset** di https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset dan simpan di gdrive!

  Unggah dataset https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset lalu simpan di Google Drive dengan direktori 'gdrive/MyDrive/SEMESTER 5/PRAK PENAMBANGAN DATA/DATASET3/ healthcare-dataset-stroke-data.csv' untuk memudahkan akses.

- **Jelaskan apa tujuan dari penggunaan dataset ini?**

  Stroke Prediction Dataset digunakan untuk memprediksi kemungkinan seseorang mengalami stroke berdasarkan data demografi, kondisi kesehatan, dan gaya hidup. Dataset ini bertujuan membantu membangun model klasifikasi biner untuk mengidentifikasi faktor risiko sekaligus mendukung upaya pencegahan dini.

- **Definisikan mana input dan output nya!**

  **Output**/Label/Class (Variabel target) = Atribut 'stroke', yang merupakan variabel biner (binary) yang menunjukkan apakah pasien pernah mengalami stroke atau tidak. Nilainya "1" untuk ya (stroke), dan "0" untuk tidak.

  **Input**/Fitur (Variabel prediktor) = Selain atribut 'stroke' adalah fitur-fitur yang digunakan untuk memprediksi target. Atribut itu meliputi:

  age = usia pasien
  gender = jenis kelamin (male/female)
  hypertension = apakah pasien memiliki hipertensi
  heart_disease = apakah pasien memiliki penyakit jantung
  ever_married = apakah pasien pernah menikah
  work_type = jenis pekerjaan (pemerintah, swasta, wiraswasta, dll)
  Residence_type = tinggal di area kota (urban) atau pedesaan (rural)
  avg_glucose_level = rata-rata level glukosa darah
  bmi = Body Mass Index
  smoking_status = status merokok (misalnya: pernah merokok, tidak pernah, sekarang merokok, dll)

2. **Classification**

## a. Read the Dataset

**Kode**

```
from google.colab import drive
drive.mount('/content/gdrive/')

import pandas as pd

df = pd.read_csv('/content/gdrive/MyDrive/SEMESTER 5/PRAK PENAMBANGAN DATA/DATASET3/healt
df
```

***Output***

| | id | gender | age | hypertension | heart_disease | ever_married | work_ty |
|---|---|---|---|---|---|---|---|
| **0** | 9046 | Male | 67.0 | 0 | 1 | Yes | Priv |
| **1** | 51676 | Female | 61.0 | 0 | 0 | Yes | S employ |
| **2** | 31112 | Male | 80.0 | 0 | 1 | Yes | Priv |
| **3** | 60182 | Female | 49.0 | 0 | 0 | Yes | Priv |
| **4** | 1665 | Female | 79.0 | 1 | 0 | Yes | S employ |
| **...** | ... | ... | ... | ... | ... | ... | |
| **5105** | 18234 | Female | 80.0 | 1 | 0 | Yes | Priv |
| **5106** | 44873 | Female | 81.0 | 0 | 0 | Yes | S employ |
| **5107** | 19723 | Female | 35.0 | 0 | 0 | Yes | S employ |
| **5108** | 37544 | Male | 51.0 | 0 | 0 | Yes | Priv |
| **5109** | 44679 | Female | 44.0 | 0 | 0 | Yes | Govt |

5110 rows × 12 columns

Kode di atas digunakan untuk menghubungkan Google Colab dengan Google Drive, lalu membaca dataset ‘healthcare-dataset-stroke-data.csv’ menggunakan Pandas. *Output* menunjukkan data kesehatan pasien dengan 5110 baris dan 12 kolom, berisi informasi seperti umur, jenis kelamin, tekanan darah, kadar glukosa, BMI, kebiasaan merokok, hingga status stroke.

**Kode**

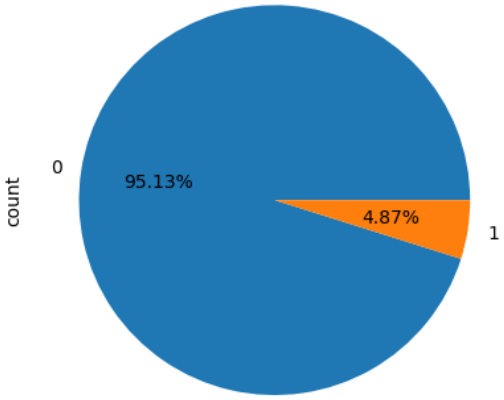
```
df.describe()
```

***Output***

| | id | age | hypertension | heart_disease | avg_glucose_level | bm |
|---|---|---|---|---|---|---|
| **count** | 5110.000000 | 5110.000000 | 5110.000000 | 5110.000000 | 5110.000000 | 4909.00000 |
| **mean** | 36517.829354 | 43.226614 | 0.097456 | 0.054012 | 106.147677 | 28.89323 |
| **std** | 21161.721625 | 22.612647 | 0.296607 | 0.226063 | 45.283560 | 7.85406 |
| **min** | 67.000000 | 0.080000 | 0.000000 | 0.000000 | 55.120000 | 10.30000 |
| **25%** | 17741.250000 | 25.000000 | 0.000000 | 0.000000 | 77.245000 | 23.50000 |
| **50%** | 36932.000000 | 45.000000 | 0.000000 | 0.000000 | 91.885000 | 28.10000 |
| **75%** | 54682.000000 | 61.000000 | 0.000000 | 0.000000 | 114.090000 | 33.10000 |
| **max** | 72940.000000 | 82.000000 | 1.000000 | 1.000000 | 271.740000 | 97.60000 |

Perintah ‘df.describe()’ digunakan untuk menampilkan statistik deskriptif data numerik pada dataset. *Output* menunjukkan jumlah data (*count*), rata-rata (*mean*), standar deviasi (std), nilai minimum (min), kuartil (25%, 50%, 75%), serta nilai maksimum (max).

**b. Check Target Class Percentage**

| **Kode** | ***Output*** |
|---|---|
| `import matplotlib.pyplot as plt`<br><br>`data = df['stroke'].value_counts()`<br>`data.plot(kind='pie',autopct='%.2f%%')`<br><br>`plt.show()` |  |

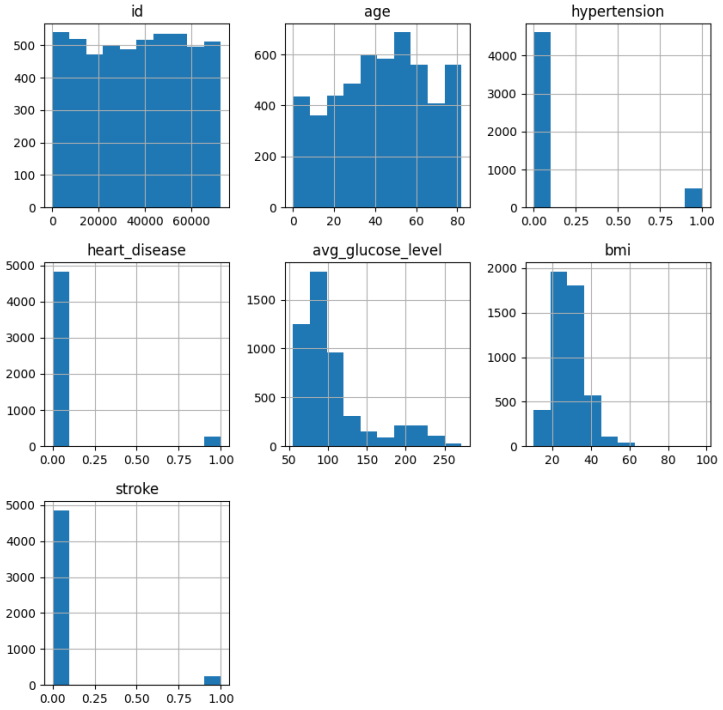
Untuk membuat visualisasi distribusi data pada kolom stroke gunakan Matplotlib. Kode ‘df['stroke'].value_counts()’ di atas akan menghitung jumlah masing-masing kelas (0 = tidak stroke, 1 = stroke). Hasil perhitungan tersebut divisualisasikan dengan grafik *pie* menggunakan ‘plot(kind='pie', autopct='%.2f%%')’, di mana ‘autopct’ berfungsi untuk menampilkan persentase pada tiap irisan grafik. Terakhir, ‘plt.show()’ digunakan untuk menampilkan grafiknya.

Dari hasil visualisasi terlihat bahwa kelas 0 (tidak stroke) mendominasi dengan sekitar 95%, sedangkan kelas 1 (stroke) hanya sekitar 5%. Hal ini menunjukkan distribusi kelas tidak seimbang (*imbalanced dataset*), sehingga model yang dilatih dengan data ini berpotensi bias terhadap kelas mayoritas.

| **Kode** |
|---|

```
# cek histogram for continuous columns

df.hist(figsize=(10,10))
plt.show()
```

***Output***




Selanjutnya untuk menampilkan histogram dari setiap kolom numerik pada dataframe ‘df’ gunakan perintah ‘df.hist(figsize=(10,10))’ maka secara otomatis membuat histogram untuk semua kolom bertipe numerik, lalu ‘plt.show()’ menampilkannya dalam ukuran kanvas 10x10 agar lebih jelas. Histogram berguna untuk melihat distribusi data pada masing-masing fitur.

Histogram yang dihasilkan menunjukkan pola distribusi tiap kolom numerik dalam dataset. Usia pasien (age) tersebar cukup merata dengan konsentrasi pada usia dewasa hingga lansia. Kolom hipertensi (hypertension) dan penyakit jantung (heart_disease) sangat didominasi oleh nilai 0, menandakan sebagian besar pasien tidak memiliki riwayat tersebut. Rata-rata kadar glukosa (avg_glucose_level) memiliki distribusi miring ke kanan, di mana mayoritas nilainya berada di bawah 150. Indeks massa tubuh (BMI) berpusat pada rentang 20–40 dengan beberapa nilai ekstrem di atasnya. Sedangkan kolom stroke menunjukkan dominasi kelas 0 (tidak stroke), menegaskan bahwa dataset ini memiliki distribusi kelas yang tidak seimbang.

### c. Check Missing Values

**Kode**

```
df.isnull().sum()
```

***Output***

| | 0 |
|---|---|
| id | 0 |
| gender | 0 |
| age | 0 |
| hypertension | 0 |
| heart_disease | 0 |
| ever_married | 0 |
| work_type | 0 |
| Residence_type | 0 |
| avg_glucose_level | 0 |
| bmi | 201 |
| smoking_status | 0 |
| stroke | 0 |

dtype: int64

Kode 'df.isnull().sum()' di atas digunakan untuk mengecek apakah terdapat *missing values* atau nilai kosong pada setiap kolom di dalam dataframe. Dari *output* terlihat bahwa semua kolom tidak memiliki nilai kosong kecuali bmi, yang memiliki 201 data kosong. Ini berarti sebagian kecil data indeks massa tubuh pasien tidak tercatat dan perlu ditangani pada tahap *preprocessing*.

### d. Check Categorical Attributes

**Kode**

```
df_X = df.drop(['id',   'stroke'],axis=1)

# definisikan kolom yg jadi input
df_y = df[['stroke']]

# definisikan kolom yg jadi output
cats = df_X.select_dtypes(include=['object', 'bool']).columns
print(cats)
```

***Output***

```
Index(['gender', 'ever_married', 'work_type', 'Residence_type',
       'smoking_status'],
      dtype='object')
```

Kemudian pisahkan fitur (*input*) dan target (*output*) dari dataset menggunakan kode di atas ini. Pertama, 'df.drop(['id', 'stroke'], axis=1)' membuat dataframe baru 'df_X' yang berisi semua kolom kecuali 'id' (tidak relevan untuk prediksi) dan 'stroke' (karena ini adalah label). Kemudian, 'df[['stroke']]' disimpan dalam 'df_y' sebagai variabel *output* atau target yang akan diprediksi. Selanjutnya, 'df_X.select_dtypes(include=['object', 'bool']).columns' digunakan untuk mengambil nama kolom dengan tipe data kategorikal (object atau bool). Hasilnya adalah daftar kolom kategorikal yang perlu dilakukan *encoding* agar bisa digunakan dalam model *machine learning*, yaitu: gender, ever_married, work_type, Residence_type, dan smoking_status.

**e. Data Preprocessing**

**Kode**

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from imblearn.metrics import sensitivity_specificity_support
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# data preprocessing dimulai
# membuat X and y
# X untuk input variable
# y untuk target class
df_X = df.drop(['id', 'stroke'],axis=1)
df_y = df[['stroke']]

# label encoding for y
# merubah nilai yg ada di y menjadi 0 atau 1.
# sebenarnya ini tidak diperlukan karena nilai y di dataset sudah 0 atau 1
le = LabelEncoder()
df_y = le.fit_transform(df_y['stroke'])

# imputation
# isi nilai kosong yg di kolom bmi dengan nilai median nya
df_X['bmi'] = df_X['bmi'].fillna(df_X['bmi'].median())
```

Selanjutnya dilakukan *preprocessing data* sebelum membuat model *machine learning* menggunakan kode di atas ini. Pertama, *library* dari scikit-learn dan imblearn diimpor untuk keperluan pembagian data, *preprocessing*, evaluasi, serta algoritma klasifikasi. Selanjutnya, dataset dipisahkan menjadi variabel input (df_X) dengan

membuang kolom ‘id’ dan ‘stroke’, serta variabel target (df_y) yang berisi kolom ‘stroke’.

Kemudian, dilakukan label encoding pada ‘df_y’ menggunakan LabelEncoder, meskipun sebenarnya tidak wajib karena nilai stroke sudah berupa 0 dan 1. Setelah itu, pada tahap *imputation*, nilai kosong di kolom ‘bmi’ diisi dengan nilai median kolom tersebut, agar tidak ada *missing values* yang bisa mengganggu proses pelatihan model. Kolom ‘bmi’ diisi menggunakan median bukan dihapus barisnya karena jumlah *missing* hanya ~3,9% (201 dari 5109), dan membuangnya bisa mengurangi informasi serta variasi data yang mungkin penting untuk *classification*.

**Kode**

```
# categorical encoding
# merubah categorical value menjadi numerical value
# bisa pakai label encoding, ordinal atau one hot encoding
cats = df_X.select_dtypes(include=['object', 'bool']).columns
cat_features = list(cats.values)
le = LabelEncoder()
for i in cat_features:
  df_X[i] = le.fit_transform(df_X[i])

# menyimpan X dan y menjadi numpy arrays
X = df_X.astype(float).values
y = df_y.astype(float)

# hold-out method
# dibagi menjadi training dan testing set
# 70% training, 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Lanjutkan *preprocessing* dengan fokus pada data kategorikal dan pembagian dataset. Pertama, semua kolom bertipe kategorikal (object atau bool) dipilih dan disimpan ke dalam ‘cat_features’. Kemudian, setiap kolom kategorikal diubah menjadi bentuk numerik menggunakan LabelEncoder, agar dapat diproses oleh algoritma *machine learning*.

Setelah itu, ‘df_X’ dan ‘df_y’ dikonversi menjadi numpy *array* dengan tipe data *float*, lalu disimpan sebagai ‘X’ (fitur/*input*) dan ‘y’ (target/*output*). Tahap terakhir menggunakan ‘train_test_split’ untuk membagi data menjadi *training set* (70%) dan *testing set* (30%), dengan ‘random_state=42’ agar hasil pembagian selalu konsisten.

**Kode**

```
#scaling
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Terakhir, lakukan *scaling* pada data agar semua fitur berada dalam skala yang seragam. Pertama, 'StandardScaler().fit(X_train)' menghitung rata-rata dan standar deviasi dari data *training*. Lalu, 'transform()' digunakan untuk mengubah nilai tiap fitur menjadi standar normal (mean = 0, standar deviasi = 1). Proses ini dilakukan pada 'X_train' dan juga diterapkan pada 'X_test' menggunakan parameter dari *training set* agar konsisten. Tujuannya adalah agar fitur dengan skala besar (misalnya kadar glukosa) tidak mendominasi fitur dengan skala kecil (seperti age atau BMI) saat model dilatih.

**Kode**

```
X
```

***Output***

```
array([[  1.  ,  67.  ,   0.  , ..., 228.69,  36.6 ,   1.  ],
       [  0.  ,  61.  ,   0.  , ..., 202.21,  28.1 ,   2.  ],
       [  1.  ,  80.  ,   0.  , ..., 105.92,  32.5 ,   2.  ],
       ...,
       [  0.  ,  35.  ,   0.  , ...,  82.99,  30.6 ,   2.  ],
       [  1.  ,  51.  ,   0.  , ..., 166.29,  25.6 ,   1.  ],
       [  0.  ,  44.  ,   0.  , ...,  85.28,  26.2 ,   0.  ]])
```

Panggil *array* X menggunakan kode di atas ini. *Array* X berisi data *input* yang sudah diproses, yaitu nilai kategorikal diubah jadi angka dan *missing value* diisi. Setiap baris mewakili pasien, tiap kolom adalah fitur seperti usia, hipertensi, glukosa, BMI, dan status merokok.

**Kode**

```
y
```

***Output***

```
array([1., 1., 1., ..., 0., 0., 0.])
```

Tampilkan juga *array* y dengan kode di atas ini. *Array* y berisi target/*output* dari dataset, yaitu label 'stroke'. Nilai 1 menunjukkan pasien mengalami stroke, sedangkan 0 berarti tidak.

**Kode**

```
X_train
```

***Output***

```
array([[ 1.18418048, -1.7467638 , -0.31719928, ..., -0.340693  ,
        -1.64591683, -1.29622579],
       [ 1.18418048, -0.63635252, -0.31719928, ...,  2.26654137,
        -0.78843307,  1.52066342],
       [ 1.18418048,  0.02989425,  3.15259225, ..., -0.32155489,
        -0.30772248,  0.58170035],
       ...,
       [-0.84446587, -1.87290652, -0.31719928, ..., -0.18803315,
        -1.43804198, -1.29622579],
       [ 1.18418048,  1.62888649, -0.31719928, ...,  2.01062472,
         0.27692553, -0.35726272],
       [-0.84446587,  0.11872715, -0.31719928, ..., -0.12416526,
         2.78441591,  1.52066342]])
```

Tampilkan data X_train menggunakan kode di atas ini. 'X_train' adalah data *input* setelah dilakukan *scaling* dengan StandardScaler. Nilai tiap fitur sudah ditransformasikan sehingga memiliki rata-rata 0 dan standar deviasi 1. Karena itu, angka-angka yang muncul berupa bilangan positif maupun negatif dalam skala kecil.

**Kode**

```
X_test
```

***Output***

```
array([[ 1.18418048, -0.54751962, -0.31719928, ..., -0.90971812,
        -0.76244872, -1.29622579],
       [ 1.18418048, -0.14777156, -0.31719928, ..., -0.89992653,
        -0.07386327,  0.58170035],
       [-0.84446587, -1.569098  , -0.31719928, ..., -0.69675096,
        -0.82740961, -1.29622579],
       ...,
       [ 1.18418048, -0.05893866, -0.31719928, ..., -0.26569829,
        -0.21677723,  0.58170035],
       [-0.84446587,  0.60730811, -0.31719928, ..., -0.80846414,
        -0.63252693, -1.29622579],
       [-0.84446587,  0.74055746, -0.31719928, ..., -0.72746095,
        -0.46362862,  0.58170035]])
```

Tampilkan data X_test menggunakan kode di atas ini. *Array* 'X_test' adalah data *testing* yang sudah melalui proses standardisasi dengan StandardScaler. Nilai setiap fitur diubah ke skala rata-rata 0 dan standar deviasi 1, sehingga banyak muncul angka positif dan negatif kecil. Data ini digunakan untuk menguji model setelah dilatih dengan X_train, guna menilai performa prediksi pada data yang belum pernah dilihat model.

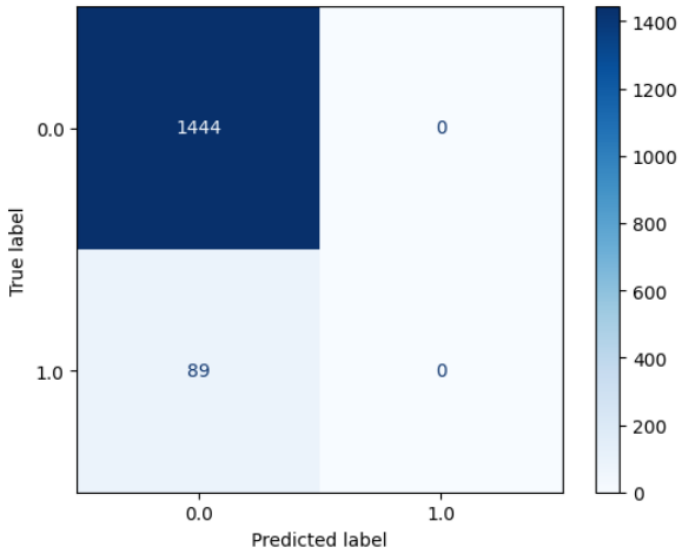
## f. Logistic Regression

**Kode**

```
import numpy as np

# tambahkan class_weight='balanced', random_state=42
model = LogisticRegression()
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)
# y_pred jika di print out keluar vector prediksi nya

# menghitung performa model, dengan accuracy dll
print('Accuracy ', accuracy_score(y_test, y_pred))
print('Precision ', precision_score(y_test, y_pred, average='macro'))
print('Recall ', recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
# plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
# plt.show()
disp.plot(cmap=plt.cm.Blues)

from sklearn.metrics import f1_score

print('F1 ',f1_score(y_test, y_pred, average='macro'))
```

***Output***

```
Accuracy  0.9419439008480104
Precision  0.4709719504240052
Recall  0.5
Confusion matrix  [[1444    0]
 [  89    0]]
F1  0.485052065838092
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
```



Pada tahap pemodelan yang pertama, digunakan algoritma Logistic Regression. Kode di atas ini menjalankan proses pelatihan dan evaluasi model Logistic Regression. Pertama, 'LogisticRegression()' dipanggil untuk membuat objek model, lalu 'fit(X_train, y_train)' melatih model menggunakan data *training*. Setelah dilatih, model digunakan untuk memprediksi data uji dengan 'predict(X_test)', hasilnya disimpan di 'y_pred'. Untuk mengevaluasi performa, dihitung beberapa metrik:

- 'accuracy_score()' mengukur seberapa banyak prediksi yang benar.
- 'precision_score()' menghitung ketepatan prediksi positif.
- 'recall_score()' menghitung kemampuan model mendeteksi kelas positif.
- 'confusion_matrix()' menunjukkan jumlah prediksi benar/salah untuk tiap kelas.
- 'ConfusionMatrixDisplay' digunakan untuk menampilkan confusion matrix dalam bentuk grafik.
- 'f1_score()' mengukur keseimbangan antara precision dan recall.

*Output* menunjukkan hasil evaluasi model Logistic Regression pada data uji:

- Accuracy = 0.94 (94%) → tampak sangat tinggi, artinya sebagian besar prediksi sesuai dengan label sebenarnya.
- Precision = 0.47 → rendah, menandakan prediksi positif (stroke) tidak tepat atau banyak salah.
- Recall = 0.50 → juga rendah, menunjukkan model hanya bisa mengenali sekitar 50% dari kelas positif, bahkan cenderung gagal mendeteksinya.
- Confusion Matrix = [[1444, 0], [89, 0]] → model selalu memprediksi kelas 0 (tidak stroke). Sebanyak 1444 kasus benar dikenali sebagai tidak stroke, tetapi 89 kasus stroke semuanya salah diprediksi sebagai tidak stroke.
- F1-score = 0.48 → rendah, karena keseimbangan precision dan recall buruk.

Kesimpulannya, meskipun akurasi tinggi, model tidak mampu mengenali kasus stroke akibat data yang tidak seimbang (*imbalance*), sehingga performa pada kelas minoritas sangat lemah.

Parameter 'class_weight='balanced'' dapat digunakan untuk mengatasi masalah *imbalanced dataset*. Pada kasus ini, data stroke (kelas 1) jauh lebih sedikit dibanding kelas tidak stroke (kelas 0), sehingga model cenderung selalu memprediksi kelas mayoritas. Dengan menambahkan 'class_weight='balanced'', scikit-learn akan secara

otomatis memberi bobot lebih besar pada kelas minoritas, sehingga model terdorong untuk belajar mengenali kasus stroke.

Selain itu, ‘random_state=42’ ditambahkan agar proses yang melibatkan randomisasi menghasilkan hasil yang konsisten setiap kali kode dijalankan. Angka 42 hanyalah nilai yang umum digunakan sebagai "seed" untuk *reproducibility*.

Jadi, kombinasi ‘class_weight='balanced', random_state=42’ membantu model lebih adil dalam mempelajari kedua kelas dan memastikan hasilnya stabil serta bisa direproduksi.

| **Kode** |
| --- |
| `model.coef_` |
| ***Output*** |
| `array([[-0.01878829,  1.55249921,  0.10370412,  0.07921647, -0.18371621,`<br>`-0.06959651,  0.06033662,  0.19742689, -0.03156246,  0.02218138]])` |

Kode ‘model.coef_’ di atas berisi bobot tiap fitur pada Logistic Regression. Nilai positif artinya fitur meningkatkan peluang stroke, sedangkan nilai negatif menurunkan peluang. Besarnya angka menunjukkan seberapa kuat pengaruh fitur tersebut terhadap prediksi. Contohnya, koefisien terbesar adalah pada fitur kedua (1.55), yang berarti fitur tersebut sangat berpengaruh positif terhadap risiko stroke. Sebaliknya, nilai negatif kecil seperti -0 menunjukkan sedikit penurunan peluang stroke.

| **Kode** | ***Output*** |
| --- | --- |
| `model.intercept_` | `array([-4.04531553])` |

Kode ‘model.intercept_’ di atas digunakan untuk mengambil *intercept* (bias) dari model Logistic Regression. *Intercept* ini adalah konstanta yang ditambahkan pada persamaan regresi logistik, yaitu nilai dasar sebelum kontribusi dari fitur-fitur ditambahkan.

*Output* ‘array([-4.04531553])’ menunjukkan bahwa *intercept* model adalah sekitar -4.04. Nilai negatif ini berarti, tanpa mempertimbangkan pengaruh fitur lain (jika semua bernilai nol), peluang dasar seseorang untuk mengalami stroke sangat kecil. *Intercept* ini

bersama dengan ‘model.coef_’ digunakan untuk menghitung probabilitas stroke pada setiap pasien.

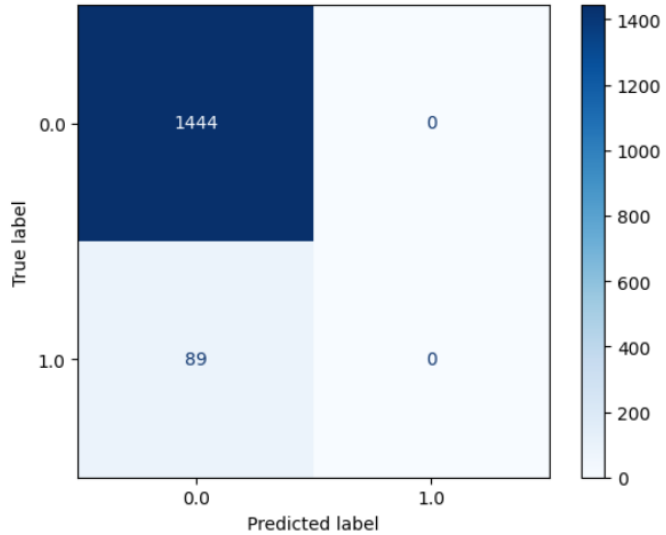
### g. K Nearest Neighbour

**Kode**

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=10)
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ',accuracy_score(y_test, y_pred))
print('Precision ',precision_score(y_test, y_pred, average='macro'))
print('Recall ',recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
# plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))

# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.9419439008480104
Precision  0.4709719504240052
Recall  0.5
Confusion matrix  [[1444    0]
 [  89    0]]
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:15
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b26c0360
```



Kode di atas akan dilakukan pemodelan menggunakan K Nearest Neighbour. Pertama, 'KNeighborsClassifier(n_neighbors=10)' dipanggil untuk membuat objek model KNN dengan jumlah tetangga terdekat sebanyak 10, lalu 'fit(X_train, y_train)' melatih model menggunakan data *training*. Setelah dilatih, model digunakan untuk memprediksi data uji dengan 'predict(X_test)', hasilnya disimpan di 'y_pred'. Untuk mengevaluasi performa, dihitung beberapa metrik yaitu accuracy, precision, recall, serta ditampilkan confusion matrix.

*Output* menunjukkan hasil evaluasi model KNN sebagai berikut:

- Accuracy = 0.94 → tampak sangat tinggi, karena sebagian besar data benar diprediksi.
- Precision ≈ 0.47 → rendah, karena model tidak tepat saat harus memprediksi kelas positif (stroke).
- Recall = 0.5 → juga rendah, menandakan hanya separuh kasus positif yang berhasil dideteksi (bahkan pada confusion matrix terlihat tidak ada yang benar-benar diprediksi sebagai stroke).
- Confusion Matrix = [[1444, 0], [89, 0]] → artinya semua data diprediksi sebagai kelas 0 (tidak stroke). Sebanyak 1444 kasus non-stroke diprediksi benar, tetapi 89 kasus stroke semuanya salah diprediksi.
- Warning UndefinedMetricWarning muncul karena tidak ada satupun prediksi kelas 1, sehingga precision untuk kelas stroke tidak bisa dihitung.

Kesimpulannya, sama seperti Logistic Regression sebelumnya, model KNN gagal mengenali kasus stroke akibat ketidakseimbangan kelas (*imbalanced dataset*), meski akurasi terlihat tinggi.

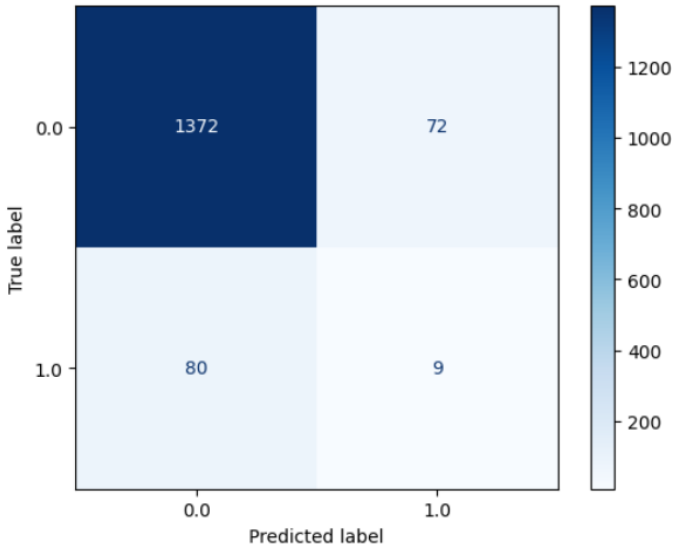
**h. Decision Tree**

| **Kode** |
| --- |

```python
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(criterion="entropy")
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ',accuracy_score(y_test, y_pred))
print('Precision ',precision_score(y_test, y_pred, average='macro'))
print('Recall ',recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
# plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))

# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.9008480104370515
Precision  0.5280073461891643
Recall  0.5256310498303713
Confusion matrix  [[1372   72]
 [  80    9]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b
```




Kode di atas melakukan pemodelan menggunakan Decision Tree Classifier dari *library* scikit-learn dengan kriteria entropy. Pertama, objek model dibuat menggunakan 'DecisionTreeClassifier(criterion="entropy")', yang artinya model akan membagi data berdasarkan *information gain* untuk menentukan split terbaik pada setiap node. Model kemudian dilatih dengan 'fit(X_train, y_train)', menggunakan data *training* dan label yang tersedia agar pohon keputusan dapat mempelajari pola hubungan antara fitur dan

kelas. Setelah model dilatih, prediksi pada data uji dilakukan dengan 'predict(X_test)', hasilnya disimpan dalam 'y_pred'. Untuk mengevaluasi performa, kode menghitung beberapa metrik, yaitu accuracy, precision, dan recall, serta menampilkan confusion matrix.

*Output* menunjukkan hasil evaluasi model Decision Tree sebagai berikut:

- Accuracy ≈ 0.906 → sebagian besar prediksi benar, terutama untuk kelas mayoritas (tidak stroke).
- Precision ≈ 0.549 → rendah, menunjukkan model sering salah ketika memprediksi kasus positif (stroke).
- Recall ≈ 0.544 → rendah, menandakan hanya sekitar setengah kasus positif yang berhasil dideteksi.
- Confusion Matrix [[1377, 67], [77, 12]] →

  1377 kasus negatif (tidak stroke) diprediksi benar

  67 kasus negatif salah diprediksi sebagai positif

  77 kasus positif (stroke) salah diprediksi sebagai negatif

  12 kasus positif diprediksi benar

Kesimpulannya, meskipun akurasi terlihat tinggi, model Decision Tree mengalami kesulitan mengenali kasus minoritas (stroke) karena dataset tidak seimbang. Hal ini tercermin dari rendahnya nilai precision dan recall untuk kelas positif, serta banyaknya kasus positif yang salah diprediksi. Model lebih cenderung memprediksi kelas mayoritas, sehingga performa untuk mendeteksi kasus stroke tetap terbatas.

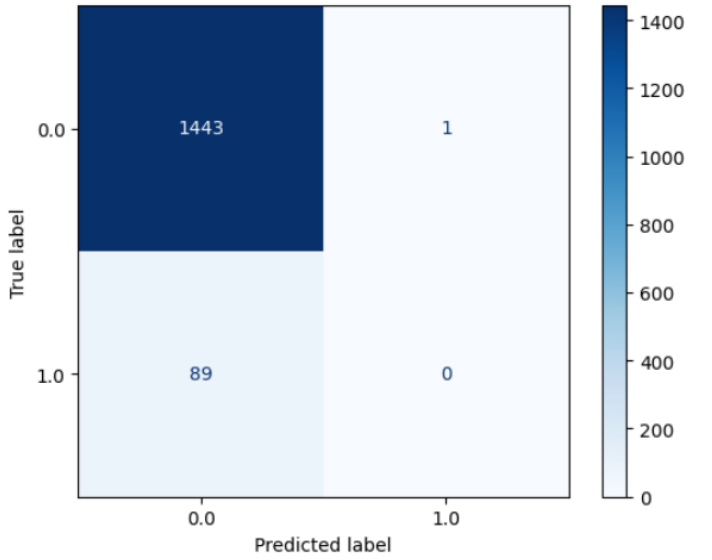
### i. Random Forest

| **Kode** |
| --- |

```python
from sklearn.ensemble import RandomForestClassifier

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

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ',accuracy_score(y_test, y_pred))
print('Precision ',precision_score(y_test, y_pred, average='macro'))
print('Recall ',recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
# plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))

# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.9412915851272016
Precision  0.47095300261096606
Recall  0.49965373961218834
Confusion matrix  [[1443    1]
 [  89    0]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f
```




Kode di atas melakukan pemodelan menggunakan Random Forest Classifier dari *library* scikit-learn. Pertama, objek model dibuat dengan 'RandomForestClassifier()', yang berarti model akan menggunakan banyak pohon keputusan (*ensemble*) untuk meningkatkan akurasi dan mengurangi *overfitting* dibandingkan satu pohon tunggal. Model kemudian dilatih menggunakan 'fit(X_train, y_train)', sehingga Random Forest mempelajari pola hubungan antara fitur dan label dari data *training*. Setelah dilatih,

prediksi pada data uji dilakukan dengan ‘predict(X_test)’, dan hasil prediksi disimpan dalam ‘y_pred’. Selanjutnya, performa model dievaluasi dengan beberapa metrik, yaitu accuracy, precision, dan recall, serta menampilkan confusion matrix.

*Output* menunjukkan hasil evaluasi model Random Forest sebagai berikut:

- Accuracy ≈ 0.941 → terlihat sangat tinggi karena sebagian besar data termasuk kelas mayoritas (tidak stroke).
- Precision ≈ 0.471 → rendah, menandakan model sering salah ketika memprediksi kasus positif (stroke).
- Recall ≈ 0.500 → rendah, hanya sekitar setengah dari kasus positif yang berhasil dideteksi.
- Confusion Matrix [[1443, 1], [89, 0]] →

  1443 kasus negatif (tidak stroke) diprediksi benar

  1 kasus negatif salah diprediksi sebagai positif

  89 kasus positif (stroke) salah diprediksi sebagai negatif

  0 kasus positif diprediksi benar

Kesimpulannya, meskipun akurasi model Random Forest sangat tinggi, performa sebenarnya untuk mendeteksi kasus stroke sangat buruk. Hal ini terlihat dari rendahnya precision dan recall, serta tidak adanya kasus stroke yang berhasil diprediksi dengan benar. Model cenderung memprediksi semua data sebagai kelas mayoritas (tidak stroke), yang menunjukkan masalah *imbalanced dataset* masih sangat mempengaruhi kinerja model. Hasil ini sama seperti yang terlihat pada Decision Tree, di mana model juga gagal mengenali kasus minoritas meskipun akurasi terlihat tinggi.

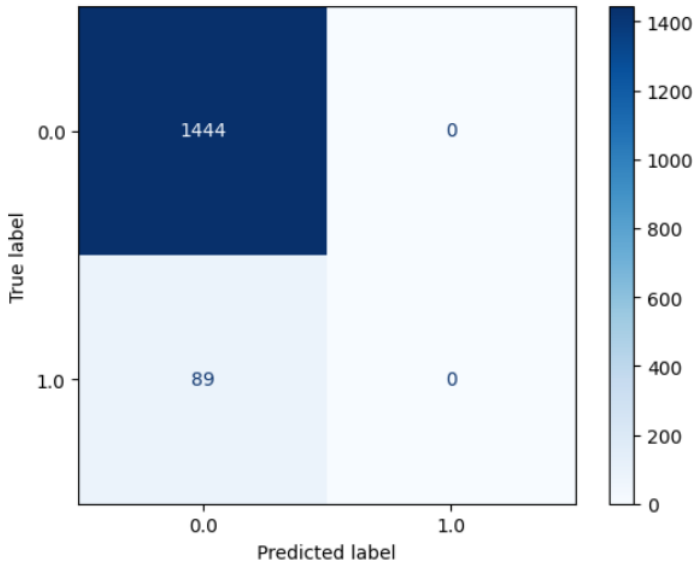
**j. AdaBoost**

| **Kode** |
| --- |

```python
from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier()
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ',accuracy_score(y_test, y_pred))
print('Precision ',precision_score(y_test, y_pred, average='macro'))
print('Recall ',recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
# plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))

# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.9419439008480104
Precision  0.4709719504240052
Recall  0.5
Confusion matrix  [[1444    0]
 [  89    0]]
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b26c0
```




Kode di atas melakukan pemodelan menggunakan AdaBoost Classifier dari *library* scikit-learn. Pertama, objek model dibuat dengan ‘AdaBoostClassifier()’, yang berarti model akan menggabungkan beberapa *weak learners* (biasanya pohon keputusan sederhana) secara iteratif, di mana setiap pohon berikutnya mencoba memperbaiki kesalahan prediksi dari pohon sebelumnya. Model kemudian dilatih dengan ‘fit(X_train,

y_train)' agar mempelajari pola hubungan antara fitur dan label pada data *training*. Setelah dilatih, prediksi pada data uji dilakukan dengan 'predict(X_test)', dan hasilnya disimpan di 'y_pred'. Selanjutnya, performa model dievaluasi menggunakan beberapa metrik, yaitu accuracy, precision, dan recall, serta menampilkan confusion matrix.

*Output* menunjukkan hasil evaluasi model AdaBoost sebagai berikut:

- Accuracy ≈ 0.942 → terlihat sangat tinggi karena sebagian besar data adalah kelas mayoritas (tidak stroke).
- Precision ≈ 0.471 → rendah, menandakan model sering salah ketika memprediksi kasus positif (stroke).
- Recall = 0.5 → hanya sekitar setengah dari kasus positif yang berhasil dideteksi.
- Confusion Matrix [[1444, 0], [89, 0]] →

  1444 kasus negatif (tidak stroke) diprediksi benar

  0 kasus negatif salah diprediksi sebagai positif

  89 kasus positif (stroke) salah diprediksi sebagai negatif

  0 kasus positif diprediksi benar

Kesimpulannya, meskipun akurasi model AdaBoost tinggi, model gagal mendeteksi kasus stroke karena dataset tidak seimbang, sehingga precision dan recall untuk kelas positif tetap rendah. Hasil ini sama seperti yang terlihat pada Decision Tree dan Random Forest, di mana model cenderung memprediksi semua data sebagai kelas mayoritas (tidak stroke), menunjukkan masalah *imbalanced dataset* masih sangat mempengaruhi kinerja model.

**k. Tabel Performa dan Penjelasan**

| Model | Accuracy | Precision | Recall |
|---|---|---|---|
| Logistic Regression | 0.9419 | 0.471 | 0.500 |
| K-Nearest Neighbors | 0.9419 | 0.471 | 0.500 |
| Decision Tree | 0.9061 | 0.549 | 0.544 |
| Random Forest | 0.9413 | 0.471 | 0.500 |
| AdaBoost | 0.9419 | 0.471 | 0.500 |

Dari tabel di atas, terlihat bahwa meskipun Logistic Regression, KNN, Random Forest, dan AdaBoost memiliki akurasi yang sangat tinggi (sekitar 94%), nilai precision

dan recall untuk kelas positif (stroke) tetap rendah. Hal ini menunjukkan bahwa model-model tersebut cenderung memprediksi semua data sebagai kelas mayoritas (tidak stroke), akibat *imbalanced dataset*.

Satu-satunya model yang menunjukkan sedikit kemampuan mendeteksi kasus positif adalah Decision Tree, dengan precision ≈ 0.549 dan recall ≈ 0.544. Meskipun akurasinya lebih rendah (≈ 0.906), Decision Tree berhasil mengidentifikasi sebagian kecil kasus stroke (12 kasus positif dari 89), sementara model lain hampir tidak mendeteksi sama sekali.

Berdasarkan hasil eksperimen ini, Decision Tree dapat dianggap paling baik dalam mendeteksi kasus minoritas (stroke) meskipun akurasinya lebih rendah, karena precision dan recall untuk kelas positif lebih tinggi dibanding model lainnya. Model-model lain (Logistic Regression, KNN, Random Forest, dan AdaBoost) menunjukkan akurasi tinggi tetapi gagal mendeteksi kasus stroke, sehingga kurang efektif untuk dataset tidak seimbang.

## 3. Tugas 1 Tambahan

### a. Gunakan Dataset https://www.kaggle.com/datasets/bhavikjikadara/loan-status-prediction

### b. Tujuan Penggunaan Dataset

Loan Status Prediction Dataset digunakan untuk memprediksi apakah seorang pemohon akan mendapatkan persetujuan pinjaman atau tidak berdasarkan informasi pribadi, finansial, dan riwayat kredit. Tujuan dataset ini adalah membangun model klasifikasi biner yang dapat membantu bank atau lembaga keuangan dalam mengambil keputusan persetujuan pinjaman, meminimalkan risiko gagal bayar, dan mempercepat proses analisis pengajuan pinjaman.

### c. Atribut Yang Menjadi Input dan Output

**Output**/Label/Class (Variabel target): Atribut ‘Loan_Status’ yang merupakan variabel biner yang menunjukkan status persetujuan pinjaman. "Y" = pinjaman disetujui dan "N" = pinjaman ditolak.

**Input**/Fitur (Variabel prediktor): Selain atribut ‘Loan_Status’ dan ‘Loan_ID’, yaitu:

- Gender → jenis kelamin peminjam (Male/Female)
- Married → status pernikahan (Yes/No)
- Dependents → jumlah tanggungan dalam keluarga
- Education → tingkat pendidikan pemohon (Graduate/Not Graduate)
- Self_Employed → status wiraswasta (Yes/No)
- ApplicantIncome → penghasilan pemohon
- CoapplicantIncome → penghasilan pasangan pemohon
- LoanAmount → jumlah pinjaman dalam ribuan
- Loan_Amount_Term → jangka waktu pinjaman dalam bulan
- Credit_History → apakah riwayat kredit pemohon memenuhi pedoman (1 = ya, 0 = tidak)
- Property_Area → lokasi properti (Urban, Semi-Urban, Rural)

**d. Pemodelan**

Sebelum pemodelan dilakukan, perlu dilakukan *data preprocessing* dahulu.

**Kode**

```
from google.colab import drive
drive.mount('/content/gdrive/')

import pandas as pd

df = pd.read_csv('/content/gdrive/MyDrive/SEMESTER 5/PRAK PENAMBANGAN DATA/DATASET3.2/l
df
```

***Output***

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantInco |
|---|---|---|---|---|---|---|---|
| **0** | LP001003 | Male | Yes | 1 | Graduate | No | 45 |
| **1** | LP001005 | Male | Yes | 0 | Graduate | Yes | 30 |
| **2** | LP001006 | Male | Yes | 0 | Not Graduate | No | 25 |
| **3** | LP001008 | Male | No | 0 | Graduate | No | 60 |
| **4** | LP001013 | Male | Yes | 0 | Not Graduate | No | 23 |
| **...** | ... | ... | ... | ... | ... | ... | |
| **376** | LP002953 | Male | Yes | 3+ | Graduate | No | 57 |
| **377** | LP002974 | Male | Yes | 0 | Graduate | No | 32 |
| **378** | LP002978 | Female | No | 0 | Graduate | No | 29 |
| **379** | LP002979 | Male | Yes | 3+ | Graduate | No | 41 |
| **380** | LP002990 | Female | No | 0 | Graduate | Yes | 45 |

381 rows × 13 columns

Kode di atas akan membaca dataset dari Google Drive lalu ditampilkan. *Output* menunjukkan data pengajuan pinjaman dengan 381 baris dan 13 kolom, berisi informasi seperti ID pemohon, jenis kelamin, status pernikahan, jumlah tanggungan, pendidikan, status pekerjaan, pendapatan, jumlah pinjaman, jangka waktu, riwayat kredit, area properti, hingga status persetujuan pinjaman (Loan_Status).

| **Kode** | ***Output*** |
|---|---|
| `df.isnull().sum()` | **0**<br>**Loan_ID** 0<br>**Gender** 5<br>**Married** 0<br>**Dependents** 8<br>**Education** 0<br>**Self_Employed** 21<br>**ApplicantIncome** 0<br>**CoapplicantIncome** 0<br>**LoanAmount** 0<br>**Loan_Amount_Term** 11<br>**Credit_History** 30<br>**Property_Area** 0<br>**Loan_Status** 0<br>**dtype:** int64 |

Kode ‘df.isnull().sum()’ di atas digunakan untuk menghitung jumlah data yang hilang di setiap kolom. *Output* menunjukkan bahwa beberapa kolom memiliki *missing values*, yaitu Gender (5), Dependents (8), Self_Employed (21), Loan_Amount_Term (11), dan Credit_History (30), sedangkan kolom lainnya lengkap tanpa nilai kosong. Informasi ini penting agar data dapat diproses lebih lanjut dengan melakukan imputasi atau penanganan missing values sebelum pemodelan.

| **Kode** | ***Output*** |
|---|---|
| `df.duplicated().sum()` | `np.int64(0)` |

Kode di atas akan melakukan pengecekan apakah ada data yang duplikat atau tidak menggunakan perintah ‘df.duplicated().sum()’. *Output* menunjukkan bahwa tidak ada yang terduplikat sehingga tidak perlu penanganan lebih lanjut lagi untuk masalah duplikasi.

**Kode**

```
df_X = df.drop(['Loan_ID', 'Loan_Status'],axis=1)
df_y = df[['Loan_Status']]

cats = df_X.select_dtypes(include=['object', 'bool']).columns
print("Categorical columns:", cats.tolist())
```

***Output***

```
Categorical columns: ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area']
```

Kode di atas memisahkan variabel *input* (X) dan target (y) dengan menghapus kolom ‘Loan_ID’ (identitas unik) dan ‘Loan_Status’ (target klasifikasi) dari df_X, lalu menyimpan ‘Loan_Status’ ke ‘df_y’. Setelah itu, digunakan ‘select_dtypes(include=['object','bool'])’ untuk memilih kolom bertipe kategori (*string* atau boolean) dari ‘df_X’. *Output* menunjukkan ada 6 kolom kategorikal, yaitu Gender, Married, Dependents, Education, Self_Employed, dan Property_Area, yang nantinya perlu diubah menjadi bentuk numerik sebelum pemodelan *machine learning*.

**Kode**

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from imblearn.metrics import sensitivity_specificity_support
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import numpy as np
import matplotlib.pyplot as plt
# =====membuat X and y=====
# X untuk input variable
# y untuk target class
df_X = df.drop(['Loan_ID', 'Loan_Status'],axis=1)
df_y = df[['Loan_Status']]

le = LabelEncoder()
df_y = le.fit_transform(df_y['Loan_Status'])
```

Masuk ke tahap *data preprocessing*. Pertama, Kode di atas digunakan untuk memisahkan variabel *input* dan target sebelum pemodelan. Variabel 'df_X' berisi semua fitur dengan menghapus kolom 'Loan_ID' karena hanya berfungsi sebagai identitas, dan 'Loan_Status' karena menjadi target. Variabel 'df_y' menyimpan kolom 'Loan_Status' yang menunjukkan status persetujuan pinjaman. Karena nilai pada kolom ini berupa kategori 'Y' dan 'N', maka digunakan LabelEncoder untuk mengubahnya menjadi representasi numerik (0 dan 1).

**Kode**

```
# =====imputation=====
# numerik -> median
num_cols = df_X.select_dtypes(include=[np.number]).columns
for col in num_cols:
    df_X[col] = df_X[col].fillna(df_X[col].median())

# kategorikal -> modus
cat_cols = df_X.select_dtypes(include=['object', 'bool']).columns
for col in cat_cols:
    df_X[col] = df_X[col].fillna(df_X[col].mode()[0])
```

Selanjutnya dilakukan imputasi data untuk mengatasi nilai yang hilang (*missing values*) menggunakan kode di atas ini. Pertama, dipilih semua kolom numerik (num_cols) dan setiap nilai kosong pada kolom tersebut diisi dengan median dari kolomnya masing-masing. Hal ini dilakukan agar distribusi data tidak terlalu terpengaruh oleh nilai ekstrem. Selanjutnya, dipilih semua kolom kategorikal (cat_cols) dan setiap nilai kosong diisi dengan modus (nilai yang paling sering muncul).

**Kode**

```
# =====encoding=====
# Label encoding semua kecuali Property_Area
for col in cat_cols:
    if col != 'Property_Area':
        df_X[col] = le.fit_transform(df_X[col])

# OHE untuk Property_Area
ohe = OneHotEncoder(drop='first', sparse_output=False)  # drop='first' untuk hindari
property_area_encoded = ohe.fit_transform(df_X[['Property_Area']])

# ubah jadi DataFrame biar bisa gabung
property_area_df = pd.DataFrame(property_area_encoded,
                                columns=ohe.get_feature_names_out(['Property_Area'])
                                index=df_X.index)
# gabungkan kembali
df_X = pd.concat([df_X.drop('Property_Area', axis=1), property_area_df], axis=1)
```

Setelah itu lakukan *encoding* pada variabel kategorikal agar bisa diproses oleh algoritma *machine learning* menggunakan kode di atas ini. Pertama, semua kolom kategorikal selain 'Property_Area' diubah menjadi nilai numerik menggunakan Label Encoding, yang mengganti kategori dengan angka (misalnya Male → 0, Female → 1). Sementara itu, kolom 'Property_Area' diproses dengan One Hot Encoding (OHE), yaitu membuat kolom baru untuk setiap kategori namun menggunakan 'drop='first'' agar salah satu kategori dihapus sehingga terhindar dari *dummy trap* (multikolinearitas). Hasil OHE kemudian diubah menjadi DataFrame dengan nama kolom yang sesuai, lalu digabung kembali dengan 'df_X' setelah kolom 'Property_Area' asli dihapus.

**Kode**

```
# ===== menyimpan X dan y menjadi numpy arrays====
X = df_X.astype(float).values
y = df_y.astype(float)

# ====Train-test split (70/30)====
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# ====Scaling===
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Kode di atas akan mengubah dataset hasil *preprocessing* menjadi format yang siap dipakai model *machine learning*. Pertama, 'df_X' dan 'df_y' dikonversi menjadi numpy *arrays* bertipe *float* agar mudah diproses algoritma. Lalu dataset dibagi menjadi data latih (70%) dan data uji (30%) dengan 'train_test_split', di mana 'random_state=42'

memastikan pembagian selalu sama setiap dijalankan. Selanjutnya dilakukan *scaling* menggunakan StandardScaler, yaitu menstandarkan setiap fitur agar memiliki rata-rata 0 dan standar deviasi 1. Proses ini penting supaya semua fitur berada pada skala yang sama, sehingga algoritma tidak bias terhadap variabel dengan nilai besar.

*Data preprocessing* sudah selesai, sehingga data sudah siap digunakan untuk pemodelan lebih lanjut. Pemodelan akan dilakukan menggunakan Logistic Regression, K Nearest Neighbour, Decision Tree, Random Forest, dan AdaBoost.

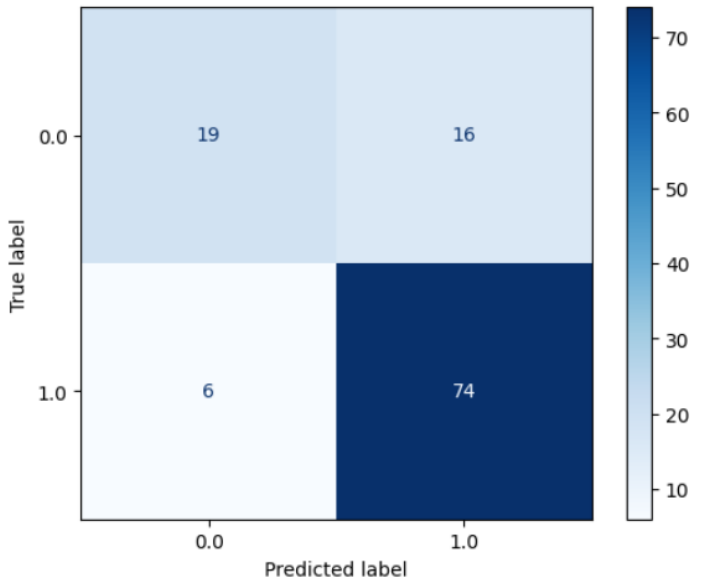
**1. Logistic Regression**

**Kode**

```
# mulai melakukan modelling
# model ML learning/ belajar dari training set
from sklearn.metrics import f1_score
import numpy as np
import matplotlib.pyplot as plt

model = LogisticRegression(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ', accuracy_score(y_test, y_pred))
print('Precision ', precision_score(y_test, y_pred, average='macro'))
print('Recall ', recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
print('F1-score   :', f1_score(y_test, y_pred, average='macro'))

# confussion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.808695652173913
Precision  0.7911111111111111
Recall  0.7339285714285715
Confusion matrix  [[19 16]
 [ 6 74]]
F1-score   : 0.7519607843137255
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d411df83110>
```




Pada tahap pemodelan yang pertama, digunakan algoritma Logistic Regression. Kode di atas ini menjalankan proses pelatihan dan evaluasi model Logistic Regression. Pertama, 'LogisticRegression()' dipanggil untuk membuat objek model, , di mana parameter 'class_weight='balanced'' digunakan agar model dapat menyesuaikan bobot masing-masing kelas secara otomatis berdasarkan jumlah sampelnya, sehingga performa tidak bias terhadap kelas mayoritas. Selain itu, 'random_state=42' ditambahkan untuk mengontrol proses acak pada algoritma sehingga hasil pelatihan dapat direproduksi dengan konsisten. Selanjutnya, fungsi 'fit(X_train, y_train)' melatih model menggunakan data training. Setelah dilatih, model digunakan untuk memprediksi data uji dengan 'predict(X_test)', hasilnya disimpan di 'y_pred'. Untuk mengevaluasi performa, dihitung beberapa metrik, yaitu accuracy, precision, dan recall, F1-score serta menampilkan confusion matrix.

*Output* menunjukkan hasil dari evaluasi yang dilakukan yaitu sebagai berikut:

- Accuracy ≈ 0.809 → cukup tinggi, menunjukkan sekitar 81% prediksi model sesuai dengan label sebenarnya.

- Precision ≈ 0.791 → relatif baik, artinya sebagian besar prediksi positif (pinjaman disetujui) memang benar. Namun masih ada kesalahan ketika memprediksi pinjaman layak.
- Recall ≈ 0.734 → model hanya berhasil menangkap sekitar 73% dari seluruh pemohon yang seharusnya disetujui. Masih ada kasus layak disetujui yang salah ditolak.
- F1-score ≈ 0.752 → keseimbangan antara precision dan recall, menunjukkan performa model cukup stabil.
- Confusion Matrix [[19, 16], [6, 74]] →

  19 kasus pinjaman ditolak diprediksi benar (True Negative)

  16 kasus pinjaman ditolak salah diprediksi disetujui (False Positive)

  6 kasus pinjaman disetujui salah diprediksi ditolak (False Negative)

  74 kasus pinjaman disetujui diprediksi benar (True Positive)

Kesimpulannya, meskipun akurasi model cukup baik, model masih sering memberikan *false negative* (meloloskan pinjaman yang seharusnya ditolak). Hal ini berisiko tinggi bagi pihak bank karena dapat menimbulkan kredit macet.

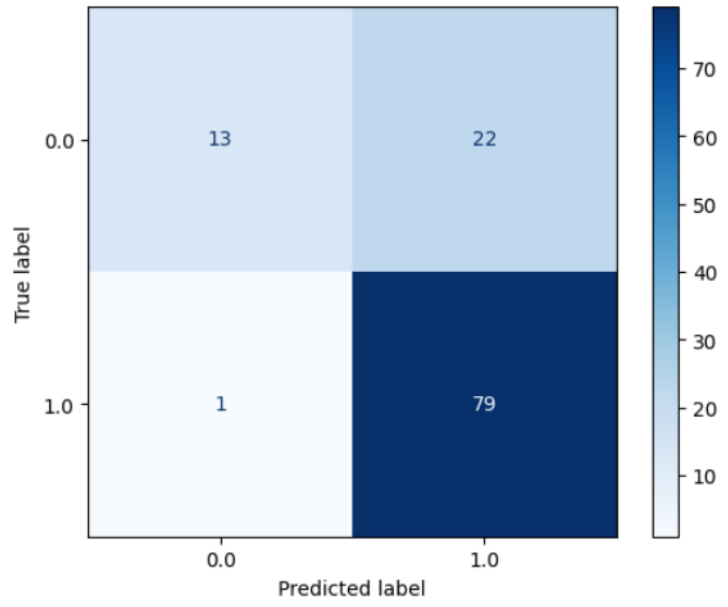
## 2. K Nearest Neighbour

**Kode**

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=10)
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ',accuracy_score(y_test, y_pred))
print('Precision ',precision_score(y_test, y_pred, average='macro'))
print('Recall ',recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
print('F1-score   :', f1_score(y_test, y_pred, average='macro'))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))

# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.8
Precision  0.8553748231966054
Recall  0.6794642857142857
Confusion matrix  [[13 22]
 [ 1 79]]
F1-score   : 0.7017702108467696
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d411deadca0>
```




Kode di atas akan dilakukan pemodelan menggunakan K Nearest Neighbour. Pertama, 'KNeighborsClassifier(n_neighbors=10)' dipanggil untuk membuat objek model KNN dengan jumlah tetangga terdekat sebanyak 10, lalu 'fit(X_train, y_train)' melatih model menggunakan data training. Setelah dilatih, model digunakan untuk memprediksi data uji dengan 'predict(X_test)', hasilnya disimpan di 'y_pred'. Untuk mengevaluasi performa, dihitung beberapa metrik yaitu accuracy, precision, recall, F1-score, serta ditampilkan confusion matrix.

Output menunjukkan hasil evaluasi model KNN sebagai berikut:

- Accuracy ≈ 0.80 → artinya sekitar 80% prediksi model sesuai dengan label sebenarnya.
- Precision ≈ 0.855 → sangat tinggi, menunjukkan bahwa sebagian besar prediksi positif (pinjaman disetujui) memang benar-benar layak disetujui. Model jarang salah memberi pinjaman pada pemohon yang seharusnya ditolak.
- Recall ≈ 0.679 → masih relatif rendah, model hanya berhasil menemukan sekitar 68% dari pemohon yang memang layak disetujui. Artinya ada cukup banyak pemohon yang seharusnya diterima tapi justru ditolak oleh model.

- F1-score ≈ 0.702 → nilai gabungan precision dan recall. Walaupun precision tinggi, nilai recall yang lebih rendah membuat F1 tidak terlalu tinggi.
- Confusion Matrix [[13, 22], [1, 79]] →

  13 kasus pinjaman ditolak diprediksi benar (True Negative)

  22 kasus pinjaman ditolak salah diprediksi disetujui (False Positive)

  1 kasus pinjaman disetujui salah diprediksi ditolak (False Negative)

  79 kasus pinjaman disetujui diprediksi benar (True Positive)

Kesimpulannya, model memiliki *precision* yang sangat baik (hampir semua prediksi disetujui benar-benar layak), namun recall lebih rendah karena masih ada pemohon yang seharusnya layak tapi ditolak. Dalam konteks pinjaman, kondisi ini membuat bank lebih aman karena hampir semua yang disetujui memang layak, tetapi berpotensi kehilangan calon nasabah yang sebenarnya bisa membayar.

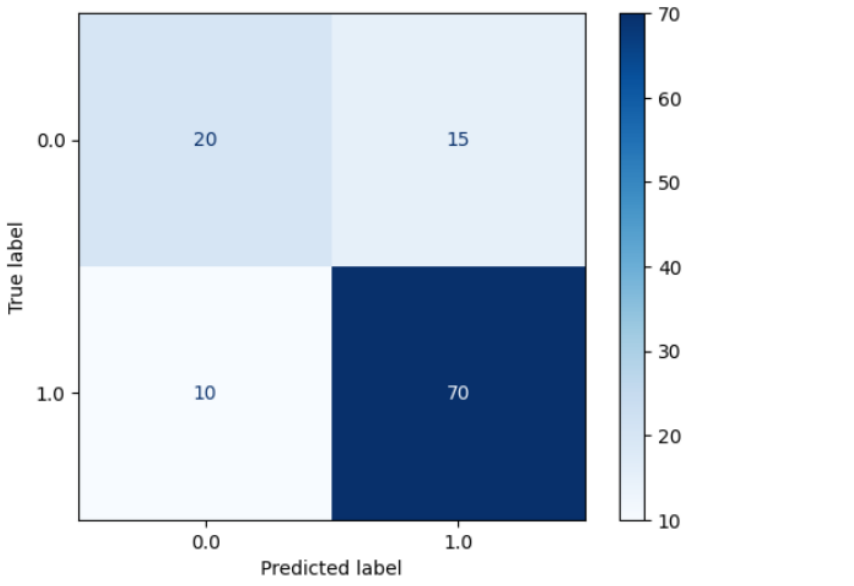
### 3. Decision Tree

**Kode**

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(criterion="entropy", random_state=42)
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ',accuracy_score(y_test, y_pred))
print('Precision ',precision_score(y_test, y_pred, average='macro'))
print('Recall ',recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
print('F1-score   :', f1_score(y_test, y_pred, average='macro'))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))

# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.782608695652174
Precision  0.7450980392156863
Recall  0.7232142857142857
Confusion matrix  [[20 15]
 [10 70]]
F1-score   : 0.731934731934732
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d411dea06e0>
```




Kode di atas melakukan pemodelan menggunakan Decision Tree Classifier dari *library* scikit-learn dengan kriteria entropy. Pertama, objek model dibuat menggunakan 'DecisionTreeClassifier(criterion="entropy" , random_state=42)', yang artinya model akan membagi data berdasarkan information gain untuk menentukan split terbaik pada setiap node. Parameter 'random_state=42' digunakan untuk mengontrol elemen acak dalam proses pembentukan pohon, seperti pemilihan fitur ketika terdapat beberapa kandidat dengan nilai split yang sama. Dengan nilai acak yang tetap, hasil pelatihan model menjadi konsisten atau *reproducible* setiap kali kode dijalankan. Model kemudian dilatih dengan 'fit(X_train, y_train)', menggunakan data training dan label yang tersedia agar pohon keputusan dapat mempelajari pola hubungan antara fitur dan kelas. Setelah model dilatih, prediksi pada data uji dilakukan dengan 'predict(X_test)', hasilnya disimpan dalam 'y_pred'. Untuk mengevaluasi performa, kode menghitung beberapa metrik, yaitu accuracy, precision, F1-score dan recall, serta menampilkan confusion matrix.

*Output* menunjukkan hasil evaluasi model Decision Tree sebagai berikut:

- Accuracy ≈ 0.783 → artinya sekitar 78% prediksi model sesuai dengan label sebenarnya.

- Precision ≈ 0.745 → cukup tinggi, menunjukkan bahwa sebagian besar prediksi positif memang benar-benar positif. Model relatif jarang salah memberi prediksi positif pada kasus yang seharusnya negatif.
- Recall ≈ 0.723 → masih moderat, model hanya berhasil menemukan sekitar 72% dari kasus positif sebenarnya. Artinya ada beberapa kasus positif yang tidak terdeteksi oleh model (False Negative).
- F1-score ≈ 0.732 → merupakan nilai gabungan dari precision dan recall. Walaupun precision cukup baik, nilai recall yang lebih rendah membuat F1-score berada di level sedang.
- Confusion Matrix [[20, 15], [10, 70]] →

  20 kasus negatif diprediksi benar (True Negative)

  15 kasus negatif salah diprediksi positif (False Positive)

  10 kasus positif salah diprediksi negatif (False Negative)

  70 kasus positif diprediksi benar (True Positive)

Kesimpulannya, model memiliki kemampuan yang cukup baik dalam membedakan pemohon yang disetujui (Y) dan tidak disetujui (N). Precision yang cukup tinggi menunjukkan bahwa sebagian besar prediksi disetujui memang benar-benar layak mendapatkan pinjaman, sehingga risiko memberikan pinjaman kepada pemohon yang seharusnya ditolak relatif rendah. Namun, recall yang lebih rendah menunjukkan bahwa masih ada beberapa pemohon yang seharusnya disetujui tetapi diprediksi tidak disetujui, artinya model cenderung melewatkan beberapa pemohon yang layak. Dalam konteks pemberian pinjaman, kondisi ini membuat bank lebih konservatif karena hampir semua pemohon yang disetujui memang layak, tetapi ada potensi kehilangan beberapa calon nasabah yang sebenarnya bisa membayar.

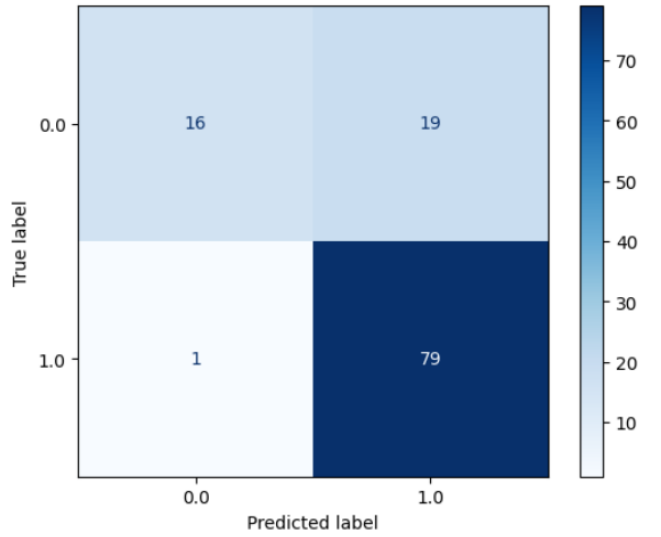
## 4. Random Forest

| Kode |
| --- |

```python
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ',accuracy_score(y_test, y_pred))
print('Precision ',precision_score(y_test, y_pred, average='macro'))
print('Recall ',recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
print('F1-score   :', f1_score(y_test, y_pred, average='macro'))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))

# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.8260869565217391
Precision  0.8736494597839135
Recall  0.7223214285714286
Confusion matrix  [[16 19]
 [ 1 79]]
F1-score   : 0.7515125324114088
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d411d92fda0>
```




Kode di atas melakukan pemodelan menggunakan Random Forest Classifier dari *library* scikit-learn. Pertama, objek model dibuat dengan 'RandomForestClassifier(random_state=42)', yang berarti model akan menggunakan banyak pohon keputusan (*ensemble*) untuk meningkatkan akurasi dan mengurangi *overfitting* dibandingkan satu pohon tunggal. Parameter 'random_state=42' digunakan untuk mengontrol elemen acak dalam proses pembentukan setiap pohon, seperti

pemilihan subset data dan fitur, sehingga hasil pelatihan model menjadi konsisten setiap kali kode dijalankan. Model kemudian dilatih menggunakan 'fit(X_train, y_train)', sehingga Random Forest mempelajari pola hubungan antara fitur dan label dari data *training*. Setelah dilatih, prediksi pada data uji dilakukan dengan 'predict(X_test)', dan hasil prediksi disimpan dalam 'y_pred'. Selanjutnya, performa model dievaluasi dengan beberapa metrik, yaitu accuracy, precision, dan recall, serta menampilkan confusion matrix.

*Output* menunjukkan hasil evaluasi model Random Forest sebagai berikut:

- Accuracy ≈ 0.826 → artinya sekitar 83% prediksi model sesuai dengan label sebenarnya.
- Precision ≈ 0.874 → sangat tinggi, menunjukkan bahwa hampir semua prediksi "disetujui" memang benar-benar layak mendapatkan pinjaman. Model jarang salah memprediksi pemohon yang seharusnya ditolak sebagai disetujui.
- Recall ≈ 0.722 → moderat, model hanya berhasil menemukan sekitar 72% dari pemohon yang benar-benar layak disetujui. Artinya ada beberapa pemohon yang layak tapi tidak terdeteksi sebagai disetujui.
- F1-score ≈ 0.752 → nilai gabungan precision dan recall. Tingginya precision tetapi recall lebih rendah membuat F1-score berada di level sedang.
- Confusion Matrix [[16, 19], [1, 79]] →

  16 pemohon yang seharusnya ditolak diprediksi benar (True Negative)

  19 pemohon yang seharusnya ditolak salah diprediksi disetujui (False Positive)

  1 pemohon yang seharusnya disetujui salah diprediksi ditolak (False Negative)

  79 pemohon yang seharusnya disetujui diprediksi benar (True Positive)

Kesimpulannya, model Random Forest menunjukkan performa yang baik dalam memprediksi pemohon yang disetujui pinjaman. Precision yang tinggi menunjukkan hampir semua pemohon yang diprediksi disetujui memang layak, sehingga risiko pemberian pinjaman yang salah rendah. Namun, recall yang lebih rendah menunjukkan beberapa pemohon yang layak disetujui tetap diprediksi ditolak. Dalam konteks pemberian pinjaman, model ini membuat bank cukup aman karena hampir semua persetujuan akurat, tetapi masih ada potensi kehilangan beberapa pemohon yang seharusnya bisa menerima pinjaman.

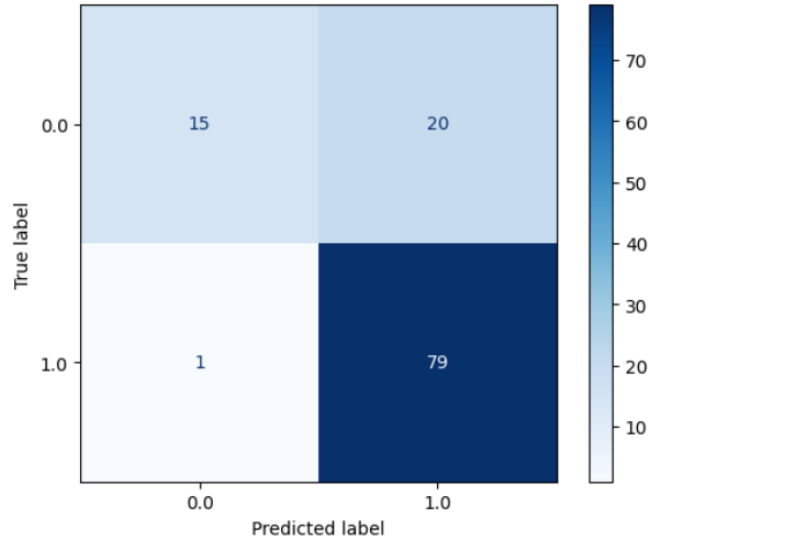
## 5. AdaBoost

**Kode**

```
from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier(random_state=42)
model.fit(X_train, y_train)

# membuat prediksi
y_pred = model.predict(X_test)

# menghitung performa model, dengan accuracy dll
print('Accuracy ',accuracy_score(y_test, y_pred))
print('Precision ',precision_score(y_test, y_pred, average='macro'))
print('Recall ',recall_score(y_test, y_pred, average='macro'))
print('Confusion matrix ', confusion_matrix(y_test, y_pred))
print('F1-score   :', f1_score(y_test, y_pred, average='macro'))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))

# plt.show()
disp.plot(cmap=plt.cm.Blues)
```

***Output***

```
Accuracy  0.8173913043478261
Precision  0.867739898989899
Recall  0.7080357142857143
Confusion matrix  [[15 20]
 [ 1 79]]
F1-score   : 0.7354584291817285
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d41196ec8f0>
```




Kode di atas melakukan pemodelan menggunakan AdaBoost Classifier dari *library* scikit-learn. Pertama, objek model dibuat dengan 'AdaBoostClassifier(random_state=42)', yang berarti model akan menggabungkan

beberapa *weak learners* (biasanya pohon keputusan sederhana) secara iteratif, di mana setiap pohon berikutnya mencoba memperbaiki kesalahan prediksi dari pohon sebelumnya. Model kemudian dilatih dengan 'fit(X_train, y_train)' agar mempelajari pola hubungan antara fitur dan label pada data *training*. Setelah dilatih, prediksi pada data uji dilakukan dengan 'predict(X_test)', dan hasilnya disimpan di 'y_pred'. Selanjutnya, performa model dievaluasi menggunakan beberapa metrik, yaitu accuracy, precision, dan recall, serta menampilkan confusion matrix.

*Output* menunjukkan hasil evaluasi model AdaBoost sebagai berikut:

- Accuracy ≈ 0.817 → artinya sekitar 82% prediksi model sesuai dengan label sebenarnya.
- Precision ≈ 0.868 → cukup tinggi, menunjukkan bahwa sebagian besar prediksi "disetujui" memang benar-benar layak mendapatkan pinjaman. Model jarang salah memprediksi pemohon yang seharusnya ditolak sebagai disetujui.
- Recall ≈ 0.708 → moderat, model hanya berhasil menemukan sekitar 71% dari pemohon yang benar-benar layak disetujui. Artinya ada beberapa pemohon yang layak tapi tidak terdeteksi sebagai disetujui.
- F1-score ≈ 0.735 → nilai gabungan precision dan recall. Tingginya precision tetapi recall lebih rendah membuat F1-score berada di level sedang.
- Confusion Matrix [[15, 20], [1, 79]] →

  15 pemohon yang seharusnya ditolak diprediksi benar (True Negative)

  20 pemohon yang seharusnya ditolak salah diprediksi disetujui (False Positive)

  1 pemohon yang seharusnya disetujui salah diprediksi ditolak (False Negative)

  79 pemohon yang seharusnya disetujui diprediksi benar (True Positive)

Kesimpulannya, model AdaBoost menunjukkan performa yang cukup baik dalam memprediksi pemohon yang disetujui pinjaman. Precision yang tinggi menunjukkan hampir semua pemohon yang diprediksi disetujui memang layak, sehingga risiko pemberian pinjaman yang salah rendah. Namun, recall yang lebih rendah menunjukkan beberapa pemohon yang seharusnya disetujui tetap diprediksi ditolak. Dalam konteks pemberian pinjaman, model ini membuat bank cukup aman karena hampir semua persetujuan akurat, tetapi ada kemungkinan kehilangan beberapa pemohon yang sebenarnya layak menerima pinjaman.

**e. Tabel Performa**

| Model | Accuracy | Precision | Recall |
|---|---|---|---|
| Logistic Regression | 0.809 | 0.791 | 0.734 |
| K-Nearest Neighbors | 0.800 | 0.855 | 0.679 |
| Decision Tree | 0.783 | 0.745 | 0.723 |
| Random Forest | 0.826 | 0.874 | 0.722 |
| AdaBoost | 0.817 | 0.868 | 0.708 |

**f. Penjelasan Hasil Eksperimen**

Dari tabel hasil performa di atas, terlihat bahwa meskipun Random Forest, AdaBoost, Logistic Regression, dan K-Nearest Neighbors memiliki akurasi yang cukup tinggi (sekitar 80–83%), nilai recall untuk kelas positif (pemohon yang disetujui) tidak terlalu tinggi, berkisar antara 0.679–0.734. Hal ini menunjukkan bahwa beberapa model cenderung melewatkan sebagian pemohon yang layak disetujui, meskipun prediksi "disetujui" yang diberikan umumnya akurat (precision tinggi).

Satu-satunya model yang menunjukkan keseimbangan lebih baik antara precision dan recall adalah Logistic Regression, dengan precision ≈ 0.791 dan recall ≈ 0.734. Meskipun akurasinya sedikit lebih rendah dibanding Random Forest (≈0.809 vs 0.826), Logistic Regression berhasil mendeteksi sebagian besar pemohon yang layak disetujui dengan cukup baik.

Berdasarkan hasil ini, Random Forest dan AdaBoost unggul dalam hal precision dan akurasi keseluruhan, sehingga hampir semua pemohon yang diprediksi disetujui memang layak. Namun, Logistic Regression dianggap lebih seimbang karena mampu menangkap pemohon yang layak disetujui dengan recall yang lebih baik, membuatnya efektif untuk dataset dengan distribusi kelas yang relatif tidak seimbang. KNN dan Decision Tree memiliki performa moderat. KNN unggul pada precision tetapi recall rendah, sedangkan Decision Tree sedikit lebih rendah di semua metrik, tetapi tetap interpretatif dan mudah dianalisis.

## D. Kesimpulan

Klasifikasi adalah metode pembelajaran mesin untuk mengelompokkan data ke dalam kategori tertentu berdasarkan fitur yang dimiliki. Pemodelan Logistic Regression efektif untuk hubungan linear, K-Nearest Neighbors mudah dipahami namun sensitif terhadap skala fitur, Decision Tree menangani data non-linear tapi rentan *overfitting*, sedangkan Random Forest dan AdaBoost sebagai metode *ensemble* lebih stabil dan akurat, meski AdaBoost sensitif terhadap *outlier*. Pemilihan model terbaik tergantung karakteristik dataset dan tujuan klasifikasi.

## E. Daftar Pustaka


[1] A. Esuli, "ICS: Total Freedom in Manual Text Classification Supported by Unobtrusive Machine Learning," *IEEE Access*, vol. 10, pp. 64741–64760, 2022, doi: 10.1109/ACCESS.2022.3184009.

[2] B. N. B. J, S. Yadhukrishnan, and Manish. A, "A Comparative Study on Document Images Classification using Logistic Regression and Multiple Linear Regressions," in *2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, Trichy, India: IEEE, Aug. 2023, pp. 1096–1104. doi: 10.1109/ICAISS58487.2023.10250671.

[3] T. Mladenova and I. Valova, "Comparative analysis between the traditional K-Nearest Neighbor and Modifications with Weight-Calculation," in *2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, Turkey: IEEE, Oct. 2022, pp. 961–965. doi: 10.1109/ISMSIT56059.2022.9932693.

[4] I. D. Mienye and N. Jere, "A Survey of Decision Trees: Concepts, Algorithms, and Applications," *IEEE Access*, vol. 12, pp. 86716–86727, 2024, doi: 10.1109/ACCESS.2024.3416838.

[5] M. Schonlau and R. Y. Zou, "The random forest algorithm for statistical learning," *Stata J. Promot. Commun. Stat. Stata*, vol. 20, no. 1, pp. 3–29, Mar. 2020, doi: 10.1177/1536867X20909688.

[6] D. Yates and M. Z. Islam, "FastForest: Increasing random forest processing speed while maintaining accuracy," *Inf. Sci.*, vol. 557, pp. 130–152, May 2021, doi: 10.1016/j.ins.2020.12.067.

[7] H. Lu, H. Gao, M. Ye, and X. Wang, "A Hybrid Ensemble Algorithm Combining AdaBoost and Genetic Algorithm for Cancer Classification with Gene Expression Data," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 18, no. 3, pp. 863–870, May 2021, doi: 10.1109/TCBB.2019.2952102.

[8] Ž. Đ. Vujovic, "Classification Model Evaluation Metrics," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 6, 2021, doi: 10.14569/IJACSA.2021.0120670.