LAPORAN FINAL PROJECT

ANALISIS KLASIFIKASI STROKE MENGGUNAKAN LOGISTIK LINEAR, ANN, KNN, RANDOM FOREST DAN CATBOOST

Disusun oleh:

Kelompok 4

Nyoman Satyawikrama Upadhana	**5025221194**
Farhan Widyanto Wibowo	**5025221209**
I Putu Raditya Partha Wijaya	**5025221210**
Muhammad Ihsan Al Khwaritsmi	**5025221211**
Putu Indra Mahendra	**5025221215**

PEMBELAJARAN MESIN A

2025

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Penyakit stroke merupakan salah satu penyebab utama kematian dan kecacatan di seluruh dunia. Stroke terjadi ketika suplai darah ke otak terganggu, yang dapat menyebabkan kerusakan jaringan otak secara permanen. Deteksi dini terhadap risiko stroke sangat penting agar dapat dilakukan intervensi yang cepat dan tepat sehingga dapat mengurangi angka mortalitas dan morbiditas.

Seiring dengan perkembangan teknologi, pendekatan berbasis kecerdasan buatan (AI) dan machine learning (ML) menjadi semakin populer dalam membantu deteksi penyakit secara otomatis. Melalui pengolahan data kesehatan, machine learning dapat mengidentifikasi pola-pola risiko yang sulit diamati oleh manusia.

Dalam proyek ini, dilakukan pengembangan dan evaluasi sistem klasifikasi deteksi penyakit stroke menggunakan berbagai model machine learning. Beberapa model yang akan dibandingkan dalam performanya adalah Logistic Regression, Neural Network, K-Nearest Neighbors (KNN), Random Forest, dan CatBoost. Melalui eksperimen ini, diharapkan dapat diperoleh model yang paling efektif dan efisien dalam memprediksi risiko stroke berdasarkan atribut-atribut kesehatan pasien.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah yang akan dibahas dalam proyek ini adalah:

1. Bagaimana membangun model machine learning untuk melakukan klasifikasi deteksi risiko stroke?
2. Bagaimana performa masing-masing model (Logistic Regression, Artificial Neural Network, KNN, Random Forest, dan CatBoost) dalam mendeteksi risiko stroke?
3. Bagaimana performa masing-masing model (Logistic Regression, Artificial Neural Network, KNN, Random Forest, dan CatBoost) menggunakan feature engineering?
4. Model manakah yang memberikan hasil terbaik dalam hal akurasi, precision, recall, dan F1-score untuk klasifikasi deteksi stroke?

1.3 Tujuan

Adapun tujuan yang ingin dicapai dalam proyek ini adalah:

1. Membangun sistem klasifikasi deteksi penyakit stroke berbasis machine learning.
2. Mengimplementasikan dan membandingkan berbagai model machine learning dalam tugas klasifikasi deteksi stroke.
3. Mengevaluasi dampak penerapan teknik *feature engineering* terhadap kinerja masing-masing model machine learning dalam klasifikasi deteksi risiko stroke
4. Menentukan model terbaik yang dapat digunakan untuk deteksi dini risiko stroke berdasarkan data kesehatan pasien.

1.4 Manfaat

Manfaat yang dapat diperoleh dari proyek ini meliputi:

1. Memberikan kontribusi dalam pengembangan teknologi deteksi dini penyakit stroke berbasis machine learning.
2. Menyediakan analisis perbandingan model untuk membantu praktisi atau peneliti dalam memilih algoritma yang tepat untuk deteksi stroke.
3. Mendukung upaya peningkatan kualitas pelayanan kesehatan dengan menyediakan alat bantu pengambilan keputusan yang berbasis data.

1.5 Batasan Masalah

Untuk menjaga fokus dan ruang lingkup penelitian, berikut adalah batasan masalah dalam proyek ini:

1. Dataset yang digunakan adalah dataset publik yang berkaitan dengan data kesehatan dan riwayat pasien terkait risiko stroke, yaitu healthcare-dataset-stroke-data.csv.
2. Fitur-fitur yang digunakan terbatas pada fitur yang tersedia dalam dataset, seperti usia, jenis kelamin, hipertensi, penyakit jantung, status merokok, dan lain-lain.
3. Model yang digunakan dalam eksperimen terbatas pada:
    - Logistic Regression
    - Neural Network
    - K-Nearest Neighbors (KNN)
    - Random Forest
    - CatBoost
4. Evaluasi model dilakukan menggunakan metrik standar klasifikasi: akurasi, precision, recall, dan F1-score.
5. Proyek ini hanya fokus pada pengembangan model prediktif, tidak mencakup pengembangan sistem implementasi di dunia nyata (seperti integrasi ke dalam sistem informasi rumah sakit).

BAB 2 METODOLOGI

2.1 Penjelasan Dataset

Menurut Organisasi Kesehatan Dunia (WHO), stroke adalah penyebab utama kematian kedua secara global, bertanggung jawab atas sekitar 11% dari total kematian. Memahami faktor-faktor risiko yang berkontribusi terhadap stroke sangat penting untuk pencegahan dan diagnosis dini.

Dataset ini dirancang secara spesifik untuk tujuan tersebut: melatih model yang mampu memprediksi kemungkinan seorang pasien mengalami stroke berdasarkan serangkaian atribut diagnostik, demografis, dan gaya hidup. Dataset ini terdiri dari 5110 sampel (baris) dan 12 atribut (kolom).

Spesifikasi dan Atribut Data

Data ini terdiri dari beberapa kolom (atribut) yang menjadi fitur (input) dan satu kolom target (output). Berdasarkan analisis pada file main.ipynb, berikut adalah rincian setiap atribut:

Kolom	Deskripsi	Nilai yang Mungkin	Tipe Data
id	Identifikasi unik untuk setiap pasien	-	Integer
gender	Jenis kelamin pasien.	"Male", "Female", "Other"	
age	Usia pasien	-	Integer
hypertension	Status hipertensi pasien	0: Pasien tidak memiliki hipertensi 1: Pasien memiliki hipertensi	Integer
heart_disease	Status penyakit jantung pasien	0: Pasien tidak memiliki penyakit jantung 1: Pasien memiliki penyakit jantung	Integer
ever_maried	Status pernikahan pasien	"No", "Yes"	Kategori (String)
work_type	Jenis pekerjaan pasien	"children",	Kategori (String)

		"Govt_jov", "Never_worked", "Private", "Self-employed"	
Residence_type	Jenis tempat tinggal pasien	"Rural", "Urban"	Kategori (String)
avg_glucose_level	Rata-rata tingkat glukosa dalam darah pasien	-	Float
bmi	Indeks massa tubuh (BMI) pasien	-	Float
smoking_status	Status merokok pasien	"formerly smoked", "never smoked", "smokes", "Unknowni"	Kategori (String)
stroke	Status apakah pasien pernah mengalami stroke	0: Pasien tidak pernah mengalami stroke 1: Pasien pernah mengalami stroke	Integer

Contoh Tampilan Data

Untuk mendapatkan gambaran awal, kita bisa melihat 5 baris pertama dari dataset menggunakan perintah data.head()

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

2.2 Desain Sistem

Untuk membangun model prediktif yang akurat, kami merancang sebuah alur kerja terstruktur yang dimulai dengan pembersihan data dan rekayasa fitur secara mendalam. Berikut alur kerja yang kami buat.

2.2.1 Tahap Preprocessing

Data mentah masih belum bisa langsung digunakan. Tahap ini bertujuan untuk membersihkan dan mentransformasi data agar sesuai untuk proses pemodelan.

2.2.1.1 Penanganan Missing Values (bmi)

```
# Find missing values
missing_bmi = df["bmi"].isna()

# Sample random values from existing BMI (with replacement)
random_bmis = df.loc[~missing_bmi, "bmi"].sample(missing_bmi.sum(), replace=True,
random_state=42)
```

```
df.loc[missing_bmi, "bmi"] = random_bmis.values
```

Untuk mengatasi hal ini, diterapkan metode Imputasi Sampel Acak (Random Sample Imputation). Metode ini bekerja dengan cara mengidentifikasi terlebih dahulu semua data bmi yang tersedia (tidak kosong) untuk membentuk sebuah distribusi empiris. Kemudian, sejumlah sampel acak ditarik dari distribusi ini dengan metode pengambilan sampel dengan pengembalian (sampling with replacement), di mana jumlah sampel yang ditarik setara dengan jumlah nilai bmi yang hilang. Nilai-nilai yang hilang tersebut kemudian digantikan oleh sampel acak yang telah ditarik. Pendekatan ini dipilih karena kemampuannya untuk menjaga distribusi statistik asli dan varians dari data bmi, sehingga tidak menimbulkan distorsi yang signifikan pada sebaran data seperti yang dapat terjadi pada metode imputasi mean atau median. Kami juga ingin tetap mempertahankan distribusi normal yang dimiliki oleh BMI.

2.2.1.2 Encoding Pada 2 Variabel Biner

```
df_encoded['ever_married'] = df_encoded['ever_married'].map({'No': 0, 'Yes': 1})
df_encoded['residence_type'] = df_encoded['residence_type'].map({'Rural': 0, 'Urban': 1})
df_encoded['gender'] = df_encoded['gender'].map({'Male': 0, 'Female': 1})
```

Metode yang digunakan adalah Manual Binary Encoding dengan memanfaatkan fungsi .map() dari library Pandas. Pendekatan ini dipilih untuk fitur-fitur dengan hanya dua kategori karena efisiensi dan kemampuannya menghasilkan representasi numerik yang jelas. Secara spesifik, tiga fitur dikonversi: ever_married (dengan pemetaan 'No' menjadi 0 dan 'Yes' menjadi 1), residence_type (dengan pemetaan 'Rural' menjadi 0 dan 'Urban' menjadi 1), dan gender (dengan pemetaan 'Male' menjadi 0 dan 'Female' menjadi 1). Proses ini mengubah fitur-fitur tersebut menjadi format biner (0 dan 1) agar bisa di proses oleh model.

2.2.1.3 Penghapusan Data Gender yang Bernilai Other

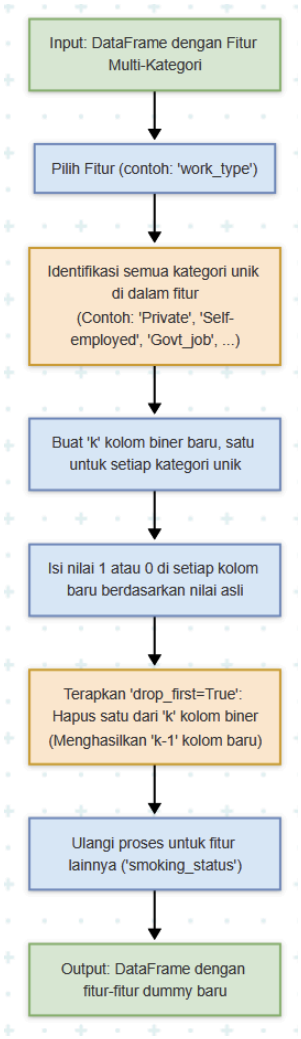
```
# Drop any 'Other' genders if still present (not enough for modeling)
df_encoded = df_encoded[df_encoded['gender'].isin([0, 1])]
```

Setelah melakukan set nilai dimana female = 1 dan male = 0. Kami melakukan penghapusan data pada kolom gender yang bernilai other, dengan alasan data yang memiliki nilai gender=’other’ hanya berjumlah 1 dari 5100 row. Data ini menurut kami tidak cukup untuk mempresentasikan sesuatu. Pada kode ini yang dilakukan, selain nilai 1 dan 0 maka row tersebut akan dihapus/drop.

2.2.1.4 Encoding Variabel Kategorikal

```
# One-hot encode multi-category variables
df_encoded = pd.get_dummies(df_encoded, columns=['work_type', 'smoking_status'],
drop_first=True)
```

Pada pra-proses ini kita menggunakan One-hot encode pada category yang memiliki lebih dari 2 nilai. Teknik ini mentransformasi setiap kategori dalam sebuah fitur menjadi kolom biner baru (dummy variable), di mana nilai 1 merepresentasikan keberadaan suatu kategori dan 0 sebaliknya. Parameter drop_first=True diterapkan untuk setiap fitur, yang bertujuan untuk mengurangi masalah multikolinearitas dengan cara menghapus satu kolom dummy dari setiap set fitur yang baru terbentuk. Sebagai hasilnya, sebuah fitur dengan k kategori akan diwakili oleh k-1 kolom biner baru. Transformasi ini memastikan bahwa data kategorikal dapat diinterpretasikan oleh model machine learning tanpa menimbulkan asumsi urutan antar kategori.



Berikut contoh sebelum dan sesudah menggunakan One-hot encode

work_type
Private
Self-employed
Private
Private
Self-employed
...
Private
Self-employed
Self-employed
Private
Govt_job

work_type_Never_worked	work_type_Private	work_type_Self-employed	work_type_children
0	1	0	0
0	0	1	0
0	1	0	0
0	1	0	0
0	0	1	0

Mari kita ambil contoh pada work_type, kolom ini memiliki nilai dengan konfigurasi seperti ini :

- Unique values in 'work_type': ['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked']
- Categories (5, object): ['Govt_job', 'Never_worked', 'Private', 'Self-employed', 'children']

Pada proses ini terlihat kolom ‘work_type_Govt_job” tidak dibuat, inilah fungsi dari parameter drop_first=True tadi. Untuk mengurangi redundant data, method get_dummies() hanya membuat k-1 fitur, dimana k adalah jumlah unik value dari kolom tersebut. Jika Never_work = 0 & Private = 0 & Self_employed = 0 & children = 0, make sudah pasti nilai Govt_job = 1. Begitu juga sebaliknya jika nilai lain selain Govt_job bernilai 1 maka Govt_job sudah dipastikan 0.

2.2.1.5 Splitting Data

Langkah fundamental dalam validasi model adalah pembagian dataset menjadi set data latih (training set) dan set data uji (testing set). Proses ini bertujuan untuk melatih model pada satu subset data dan mengevaluasi kinerjanya pada subset data lain yang belum pernah dilihat sebelumnya, sehingga memberikan estimasi performa model yang objektif.

Pertama, dataset dipisahkan menjadi matriks fitur (X), yang berisi semua variabel independen, dan vektor target (y), yang berisi variabel dependen ('stroke'). Selanjutnya, pembagian data dilakukan menggunakan fungsi train_test_split dari library Scikit-learn dengan proporsi alokasi 80% untuk data latih dan 20% untuk data uji.

Untuk memastikan bahwa distribusi kelas target ('stroke') proporsional antara set latih dan uji, digunakan metode pemisahan bertingkat (stratified splitting) dengan menyetel parameter stratify=y. Hal ini sangat krusial untuk dataset yang tidak seimbang (imbalanced) agar kedua set data memiliki representasi kelas yang sama dengan dataset asli. Selain itu, parameter random_state diatur ke nilai konstan (42) untuk menjamin pembagian data pada setiap eksekusi kode, sehingga hasil penelitian dapat diverifikasi secara konsisten.

2.2.1.6 Standarisasi Data

```
# Penskalaan Fitur
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Pada standarisasi data kami menggunakan StandardScaler dari pustaka Scikit-learn untuk mentransformasi setiap fitur sehingga memiliki rata-rata (mean) nol dan standar deviasi satu. Tujuan utama dari langkah ini adalah untuk menghilangkan bias yang mungkin timbul dari fitur-fitur yang memiliki skala atau rentang nilai yang sangat berbeda. StandardScaler dilatih hanya menggunakan data latih (X_train) untuk mempelajari parameter skala, yang kemudian diterapkan secara konsisten untuk mentransformasi baik data latih maupun data uji (X_test). Prosedur ini penting untuk mencegah kebocoran data (data leakage) dari set pengujian dan

memastikan bahwa semua fitur memberikan kontribusi yang seimbang selama proses pelatihan model. Standarisasi juga lebih cocok daripada normalisasi karena tidak sensitif terhadap outlier.

Rumus untuk standarisasi :

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

- μ (Rata-rata)
- σ (Simpangan Baku)
- x (Nilai Asli)
- z (z-score)

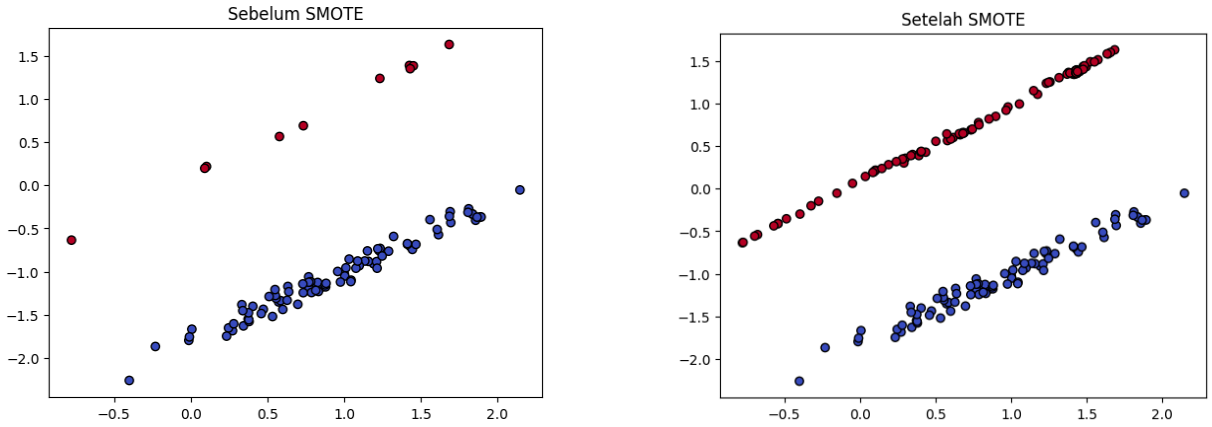
2.2.1.7 Oversampling Data

```
# Oversampling
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)
```

Setelah dilakukannya standarisasi data, kami menghadapi masalah *imbalanced dataset* yang mana sangat sering ditemukan pada dataset stroke seperti yang kami gunakan dikarenakan banyaknya kelas “0” (tidak stroke) dibandingkan “1” (stroke).

Penggunaan teknik penyeimbangan kelas menjadi sangat penting agar model tidak bias terhadap kelas mayoritas. Salah satu metode yang dapat digunakan untuk mengatasi ketimpangan ini adalah SMOTE (*Synthetic Minority Over-sampling Technique*). SMOTE merupakan metode *oversampling* yang bekerja dengan cara menghasilkan data sintetis baru pada kelas minoritas, bukan dengan menduplikasi data yang sudah ada. SMOTE bekerja dengan memilih secara acak

satu data dari kelas minoritas, kemudian mencari beberapa tetangga terdekatnya menggunakan algoritma *k-nearest neighbors*. Selanjutnya, algoritma membuat data baru dengan melakukan interpolasi antara data asli dan tetangga terdekat tersebut, menghasilkan titik-titik baru yang berada di antara keduanya dalam ruang fitur. Berikut visualisasi contoh cara kerja SMOTE.



2.2.2 Training Model dan Testing

Dalam training model dan testing, kami menggunakan 5 macam model dalam penerapannya. Semua model digunakan sebagai pembanding guna mengetahui model mana yang memberikan hasil paling efektif dalam melakukan klasifikasi stroke. Performa model diukur menggunakan metrik evaluasi, seperti Precision, Recall, dan F1 Score. Metrik evaluasi dihitung melalui rumus masing-masing.

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

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

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

TP adalah stroke terdeteksi stroke. FP adalah tidak stroke terdeteksi stroke. FN adalah stroke terdeteksi tidak stroke.

Untuk klasifikasi stroke, kami akan berfokus pada nilai Recall, dikarenakan secara medis apabila pasien sebenarnya stroke tetapi terdeteksi tidak stroke (FN) merupakan model yang berbahaya. Hal ini dikarenakan apabila pasien seharusnya mendapatkan penanganan medis, namun menjadi tidak mendapatkan karena kesalahan deteksi.

2.2.2.1 Logistik Linear

Regresi Logistik pada intinya adalah sebuah metode yang secara cerdas mengubah sebuah persamaan linear menjadi prediksi probabilitas. Proses ini dapat dipecah menjadi dua rumus kunci yang bekerja secara berurutan.

Langkah pertama dalam cara kerja Regresi Logistik adalah dengan menghitung sebuah skor agregat, yang disebut log-odds, berdasarkan semua fitur masukan. Ini dilakukan menggunakan persamaan yang identik dengan Regresi Linear. Formula ini menjumlahkan pengaruh dari setiap fitur, yang masing-masing telah dikalikan dengan bobot (koefisien) yang telah dipelajari oleh model selama proses pelatihan.

$$z = b0 + b1x1 + b2x2 + \cdots + bnxn$$

Karena skor z dari persamaan linear bukanlah probabilitas, kita memerlukan langkah kedua untuk mengubahnya menjadi nilai yang lebih bermakna. Di sinilah Fungsi Sigmoid (atau Fungsi Logistik) berperan. Fungsi ini mengambil output z yang tak terbatas dan "memetakannya" ke dalam rentang nilai yang pasti, yaitu antara 0 dan 1, yang dapat diinterpretasikan secara langsung sebagai probabilitas.

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

Setelah probabilitas p dihitung menggunakan Fungsi Sigmoid, langkah terakhir adalah membuat keputusan klasifikasi yang konkret. Ini dilakukan dengan menetapkan ambang batas (threshold), yang secara umum bernilai 0.5. Aturan keputusannya sederhana: jika nilai p yang dihitung lebih besar dari atau sama dengan 0.5, model akan memprediksi data tersebut masuk ke dalam kelas positif (kelas '1'). Jika nilai p lebih kecil dari 0.5, data tersebut akan diklasifikasikan sebagai kelas negatif (kelas '0').

```
# Latih Model Regresi Logistik
model = LogisticRegression()
model.fit(X_train_resampled, y_train_resampled)
```

Parameter yang pada Logistik Linear :

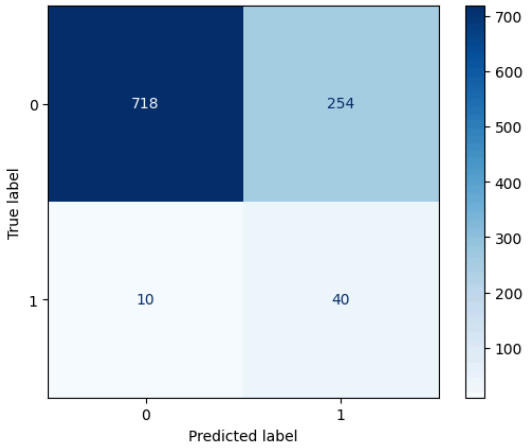
Parameter	Nilai Default	Deskripsi Singkat

penalty	l2'	Jenis regularisasi yang digunakan ('l1', 'l2', 'elasticnet', atau 'none').
dual	FALSE	Menggunakan formulasi dual atau primal. False lebih disukai saat sampel > fitur.
tol	0.0001	Toleransi atau batas eror untuk menghentikan proses training.
C	1	Kebalikan dari kekuatan regularisasi. Nilai kecil berarti regularisasi kuat.
fit_intercept	TRUE	Menentukan apakah akan menambahkan konstanta (bias atau intercept) ke model.
intercept_scaling	1	Berguna hanya jika solver='liblinear', untuk menskalakan intercept.
class_weight	None	Bobot untuk setiap kelas. None berarti semua kelas memiliki bobot 1.
random_state	None	Seed untuk pengacakan data, berguna untuk hasil yang dapat direproduksi.
solver	lbfgs'	Algoritma yang digunakan untuk optimisasi.
max_iter	100	Jumlah iterasi maksimum yang diizinkan bagi solver untuk konvergen.
multi_class	deprecated'	Cara menangani masalah multi-kelas. Default baru adalah 'auto'.
warm_start	FALSE	Jika True, model akan menggunakan solusi dari training sebelumnya sebagai titik awal.
n_jobs	None	Jumlah core CPU yang digunakan. None biasanya berarti 1 core.

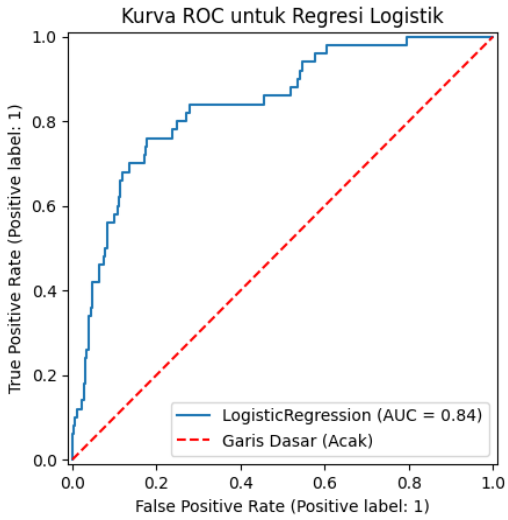
Testing

```
# Buat prediksi pada data uji
y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
```

Confusion Matrix :




```
Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.99      0.74      0.84       972
           1       0.14      0.80      0.23        50

    accuracy                           0.74      1022
```



Pada model logistic linear kami berhasil dapat skor AUC (Area Under the Curve) 0.84, yang artinya cukup baik.Terlihat pada confusion matrix, model ini berhasil 'menangkap' 40 dari 50 orang yang benar-benar stroke yang setara dengan **Recall sebesar 76%** untuk kelas 'Stroke'.

Namun, performa tinggi ini datang dengan sebuah konsekuensi signifikan yang terlihat jelas pada metrik lainnya. Model ini menghasilkan 254 kasus false positive, di mana pasien sehat salah didiagnosis sebagai penderita stroke. Akibatnya, Precision untuk kelas 'Stroke' sangat rendah, yaitu hanya 14%, yang berarti dari semua pasien yang diprediksi stroke, hanya sebagian kecil yang benar-benar menderita stroke. Secara keseluruhan, model ini telah dioptimalkan untuk

memaksimalkan deteksi kasus positif (meminimalkan false negative), namun dengan mengorbankan tingkat akurasi prediksi positifnya.

2.2.2.2 Artificial Neural Network

Neural network adalah salah satu metode dalam kecerdasan buatan yang terinspirasi dari cara kerja otak manusia. Sistem ini terdiri dari lapisan-lapisan node (disebut neuron), yang saling terhubung dan bekerja bersama untuk mengenali pola serta memproses data. Setiap neuron umumnya menerima input yang diolah dengan menggunakan fungsi aktivasi (activation function), lalu meneruskannya ke neuron-neuron pada lapisan berikutnya. Konsep neural network dapat didefinisikan sebagai berikut:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Neural network dapat digunakan dalam permasalahan regresi ataupun klasifikasi. Sehingga, merupakan model yang sangat cocok pada projek ini yaitu klasifikasi.

Dalam pelatihannya, neural network menggunakan algoritma pembelajaran seperti backpropagation untuk menyesuaikan bobot antar neuron berdasarkan seberapa besar kesalahan prediksi yang terjadi. Proses ini dilakukan berulang-ulang dengan data pelatihan untuk meningkatkan akurasi model. Model yang digunakan pada projek ini berbentuk sebagai berikut:

Layer Type	Shape	Activation function
Dense	64	relu
Dropout	64	-
Dense	32	relu
Dropout	32	-
Dense	1	sigmoid

Model yang digunakan terbentuk dari beberapa layer yang dimana terdapat layer bertipe dense dan layer dropout. Layer type digunakan untuk mempelajari pola pada neural network pada umumnya, sedangkan dropout layer digunakan untuk mencegah terjadinya overfitting agar model lebih mudah untuk generalisasi pada data yang belum pernah dipelajari. Penggunaan layer dropout umumnya akan secara random mengatur sebagian unit input menjadi 0 setiap pembaruan neural network. Neural network juga membutuhkan tahap kompilasi model untuk menentukan tiga komponen penting yaitu:

1. Optimizer : algoritma yang digunakan memperbarui bobot (weights) dari neural network untuk meminimalkan kesalahan prediksi (loss). Optimizer yang digunakan adalah Adam yang merupakan algoritma yang menggunakan konsep “momentum”, dan merupakan yang paling populer karena cepat dan stabil.

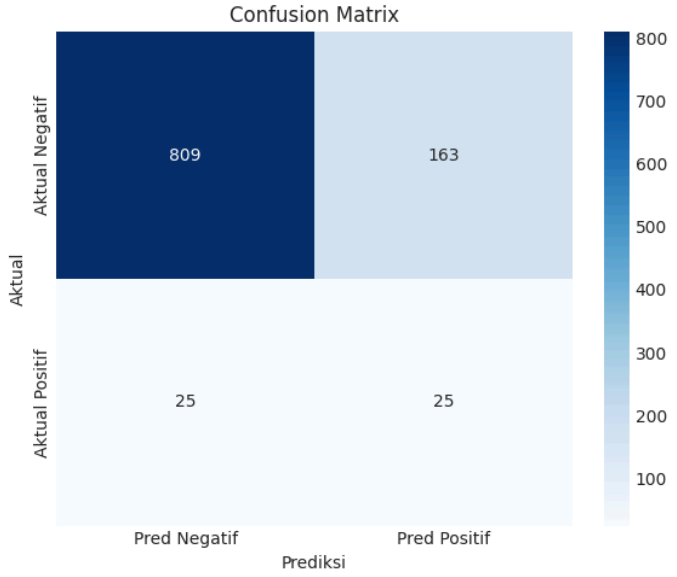
2. Loss Function : fungsi matematis yang digunakan untuk mengukur seberapa jauh prediksi model dari nilai sebenarnya. Loss function yang digunakan adalah Binary Cross Entropy yang digunakan untuk klasifikasi dua kelas.
3. Metrics : sebagai ukuran evaluasi yang digunakan untuk menilai performa model selama proses training dan testing.
   Untuk melatih neural network dari model yang telah dibuat adalah sebagai berikut:

```
history = model.fit(X_train_resampled, y_train_resampled,
                    epochs=50,
                    batch_size=32,
                    validation_data=(X_test_scaled, y_test),
                    verbose=1)
```

Parameter yang digunakan pada model neural network klasifikasi tersebut adalah sebagai berikut:

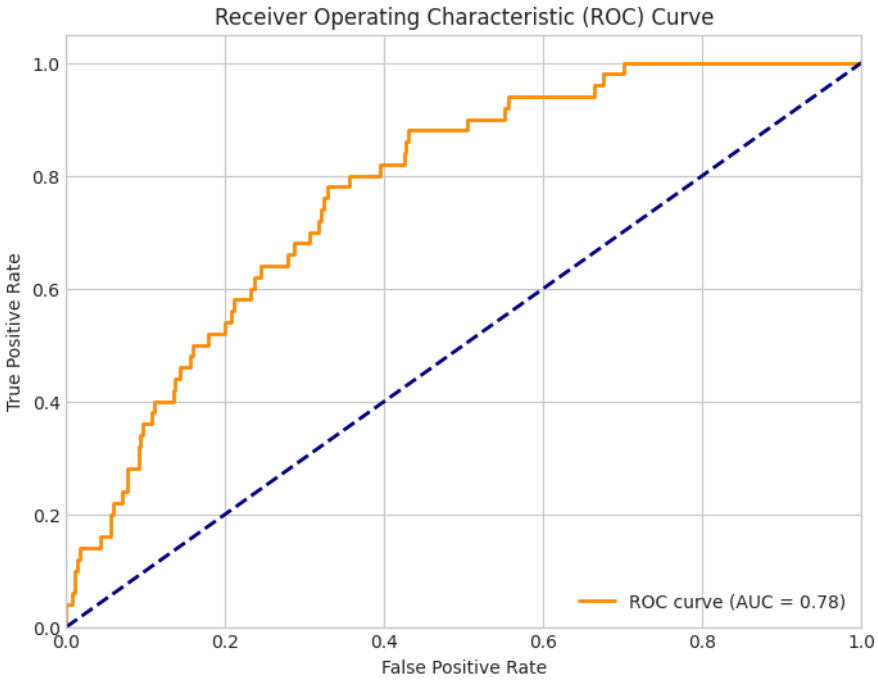
Paramater	**Nilai Default**	**Deskripsi singkat**
epochs	50	Jumlah siklus penuh pelatihan (forward + backward pass) atas seluruh data.
batch_size	32	Jumlah sampel yang diproses sebelum model melakukan update bobot.

Setelah melatih model didapatkan data hasil evaluasi sebagai berikut:



Laporan Klasifikasi:

	precision	recall	f1-score	support
0	0.97	0.83	0.90	972
1	0.13	0.50	0.21	50
accuracy			0.82	1022
macro avg	0.55	0.67	0.55	1022
weighted avg	0.93	0.82	0.86	1022



Pada evaluasi tersebut menunjukan bahwa kinerja model masih belum optimal dan masih belum bisa direkomendasikan untuk implementasi di ranah kesehatan. Meskipun analisis kurva ROC awal memberikan skor AUC yang cukup baik sebesar 0.78 dan akurasi sebesar 81.6% ,namun dengan analisis dari confusion matrix menunjukan terdapat kelemahan fundamental dimana model gagal memprediksi 50% kasus beresiko tinggi pada penyakit stroke.

2.2.2.3 KNN (K-Nearest Neighbors)

K-Nearest Neighbors adalah algoritma klasifikasi yang bekerja dengan sebuah data baru kemungkinan besar akan memiliki label yang sama dengan data-data tetangganya yang paling dekat. KNN termasuk sebagai model yang tidak memiliki proses pelatihan yang kompleks. Ketika sebuah data baru ingin diklasifikasikan, KNN akan langsung menghitung jarak antara data tersebut dengan seluruh data pelatihan menggunakan metrik tertentu, sebagai kasus ini menggunakan minkowski. Setelah seluruh jarak dihitung, algoritma akan memilih sejumlah k data terdekat (disebut tetangga) dari data baru tersebut. Proses selanjutnya adalah melakukan voting berdasarkan label kelas dari tetangga-tetangga tersebut. Kelas yang paling banyak muncul

di antara k tetangga itulah yang akan ditetapkan sebagai prediksi untuk data baru. Voting ini bisa berbobot, di mana tetangga yang lebih dekat memiliki pengaruh lebih besar dibandingkan yang lebih jauh. Pendekatan ini membuat KNN menjadi metode yang sangat intuitif dan mudah dipahami, terutama dalam kasus klasifikasi di mana hubungan antar data dapat direpresentasikan secara spasial. Berikut rumus minkowski yang digunakan sebagai default metric.

$$D(x,y) = \left(\sum_{i=0}^{n} |x_i - y_i|\right)^{p\ \frac{1}{p}}$$

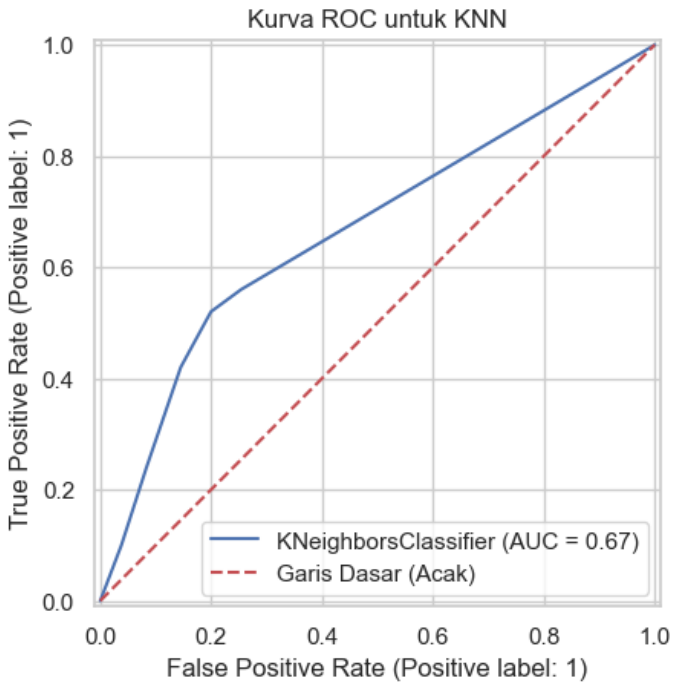
```
# Latih Model KNN
model_knn = KNeighborsClassifier(n_neighbors=5)
model_knn.fit(X_train_resampled, y_train_resampled)
```

Untuk code base, kami menggunakan parameter default yang diberikan oleh model. Berikut tabel detail parameter KNN.

Parameter	**Nilai Default**	**Deskripsi Singkat**
n_neighbors	5	Jumlah tetangga terdekat yang digunakan untuk klasifikasi. Semakin besar, semakin halus prediksi.
weights	uniform	Cara memberikan bobot ke tetangga: 'uniform' (sama rata) atau 'distance' (lebih dekat lebih berbobot).
algorithm	auto	Algoritma pencarian tetangga: 'ball_tree', 'kd_tree', 'brute', atau 'auto'.
leaf_size	30	Ukuran daun dalam struktur pohon (berpengaruh jika pakai 'kd_tree' atau 'ball_tree').
p	2	Parameter untuk metrik Minkowski: p=1 = Manhattan, p=2 = Euclidean.
metric	minkowski	Jenis metrik jarak yang digunakan, seperti 'euclidean', 'manhattan', atau 'minkowski'.
metric_params	None	Parameter tambahan untuk metrik khusus, jika diperlukan.
n_jobs	None	Jumlah core CPU yang digunakan. None berarti hanya 1 core. Gunakan -1 untuk semua core.

Setelah melatih model menggunakan parameter tersebut, berikut hasil testing yang didapat.

```
===== Hasil Evaluasi Model KNN =====

Confusion Matrix:
[[831 141]
 [ 29  21]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.97      0.85      0.91       972
           1       0.13      0.42      0.20        50

    accuracy                           0.83      1022
   macro avg       0.55      0.64      0.55      1022
weighted avg       0.93      0.83      0.87      1022


Skor AUC-ROC: 0.6668
```



Setelah dilakukan pelatihan, model KNN menunjukkan akurasi sebesar 0.83 dan AUC-ROC sebesar 0.6668. Meskipun akurasinya tinggi, performa terhadap kelas stroke (kelas 1) masih rendah, terlihat dari precision sebesar 0.13 dan f1-score hanya 0.20. Kemudian untuk confusion matrix dapat diketahui bahwa model mendeteksi true negative sebanyak 833 dan true positive 21. Sedangkan model mendeteksi false negative sebanyak 141 dan false positive 29.

2.2.2.4 Random Forest

Model *Random Forest* adalah algoritma pembelajaran mesin ensemble yang beroperasi dengan membangun ratusan *decision tree* acak selama proses pelatihan. Setiap pohon dibangun

dari sampel acak yang berbeda dari data dan subset fitur, memberikan masing-masing perspektif yang unik pada data. Untuk membuat prediksi akhir, model ini menggabungkan hasil dari semua pohon dan membuat keputusan berdasarkan suara terbanyak. Pendekatan ini-sering diibaratkan seperti meminta saran dari banyak ahli-membuat Random Forest menjadi model yang sangat akurat, stabil, dan tahan terhadap overfitting, sehingga menjadi pilihan yang tepat untuk masalah klasifikasi yang kompleks seperti prediksi risiko stroke.

```
# Latih Model Random Forest
model_rf = RandomForestClassifier(random_state=42, class_weight='balanced', n_jobs=-1)
model_rf.fit(X_train_resampled, y_train_resampled)
```

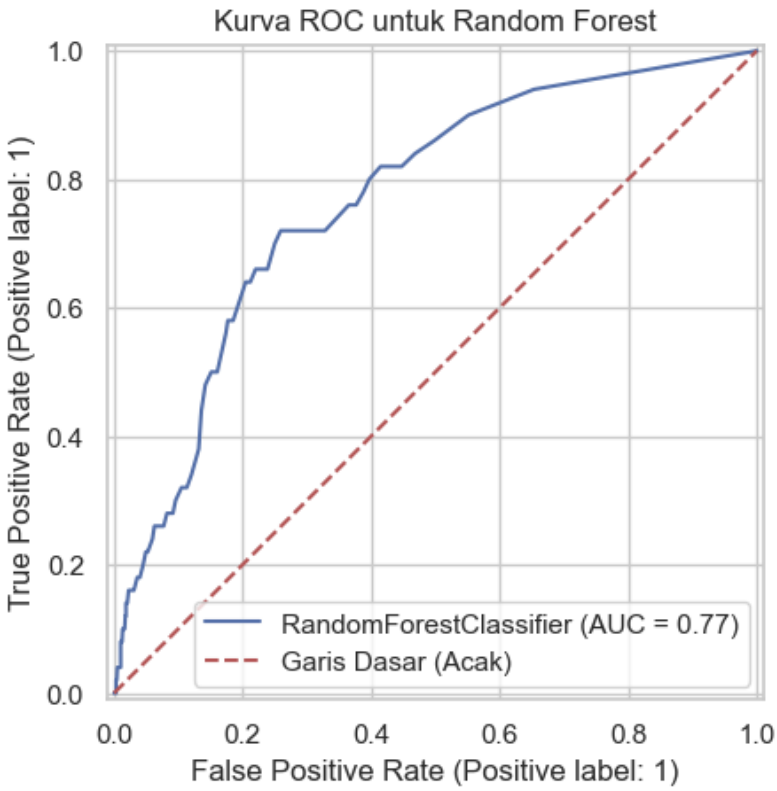
Untuk code base, kami menggunakan parameter default yang diberikan oleh model. Berikut tabel detail parameter Random Forest.

Parameter	**Nilai Default**	**Deskripsi Singkat**
n_estimators	100	Jumlah pohon keputusan (decision tree) yang akan dibangun di dalam hutan.
criterion	gini'	Fungsi untuk mengukur kualitas pemisahan pada setiap node. Pilihan lainnya adalah 'entropy'.
max_depth	None	Kedalaman maksimum dari setiap pohon. Jika None, pohon akan dikembangkan hingga semua daun murni.
min_samples_split	2	Jumlah sampel minimum yang dibutuhkan untuk dapat melakukan pemisahan pada suatu node.
min_samples_leaf	1	Jumlah sampel minimum yang harus ada di setiap ujung daun (leaf node).
max_features	sqrt'	Jumlah fitur maksimum yang akan dipertimbangkan saat mencari pemisahan terbaik di setiap node.
bootstrap	TRUE	Menentukan apakah sampel bootstrap (sampel dengan penggantian) digunakan saat membangun pohon.
random_state	None	Mengontrol keacakan. Mengaturnya ke sebuah angka (misal, 42) akan membuat hasil model dapat direproduksi.
class_weight	None	Memberikan bobot pada setiap kelas. 'balanced' sangat berguna untuk dataset tidak seimbang.
n_jobs	None	Jumlah core CPU yang digunakan untuk proses training. None berarti 1, sedangkan -1 berarti menggunakan semua core yang tersedia.

Setelah melatih model menggunakan parameter tersebut, berikut hasil testing yang didapat:

```
===== Hasil Evaluasi Model Random Forest =====

Confusion Matrix:
[[949  23]
 [ 42   8]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       972
           1       0.26      0.16      0.20        50

    accuracy                           0.94      1022
   macro avg       0.61      0.57      0.58      1022
weighted avg       0.92      0.94      0.93      1022


Skor AUC-ROC: 0.7685
```

Berikut ini merupakan kurva ROC untuk *Random Forest* setelah mendapatkan hasil testing.



Berdasarkan hasil evaluasi, model Random Forest default, meskipun mencapai akurasi keseluruhan yang tinggi yaitu 94% dan skor AUC-ROC yang cukup tinggi yaitu sekitar 0,77, pada akhirnya tidak efektif untuk tujuan utamanya yaitu mengidentifikasi pasien stroke. Akurasi yang tinggi menyesatkan karena model ini sangat baik dalam memprediksi dengan benar kelas mayoritas "tidak stroke" (dengan recall 98%), tetapi kinerjanya sangat buruk pada kelas minoritas "stroke". Kelemahan yang paling kritis adalah recall yang sangat rendah, hanya 16% untuk kasus stroke, yang berarti gagal mengidentifikasi 42 dari 50 pasien stroke yang sebenarnya dalam set tes. Hal ini membuat model ini tidak dapat diandalkan untuk aplikasi praktis atau klinis, karena model ini akan melewatkan sebagian besar individu yang berisiko.

2.2.2.5 CatBoost

CatBoost (Categorical Boosting) adalah sebuah algoritma machine learning modern yang berbasis pada gradient boosting decision trees (GBDT) dan dikembangkan oleh Yandex. Keunggulan utama yang membedakan CatBoost dari algoritma sejenisnya adalah kemampuannya yang canggih dalam menangani fitur kategorikal secara otomatis dan efisien. Alih-alih mengharuskan pengguna melakukan pra-pemrosesan manual seperti one-hot encoding, CatBoost mengimplementasikan teknik internal yang disebut ordered boosting dan permutasi untuk mengubah fitur kategorikal menjadi numerik. Pendekatan ini secara efektif mengurangi risiko overfitting dan target leakage yang sering terjadi saat menangani data kategorikal. Hasilnya, CatBoost sering kali mampu menghasilkan model dengan akurasi yang sangat tinggi

dan lebih robust, menjadikannya pilihan yang kuat untuk berbagai macam tugas klasifikasi dan regresi, terutama pada dataset yang kaya akan variabel non-numerik.

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

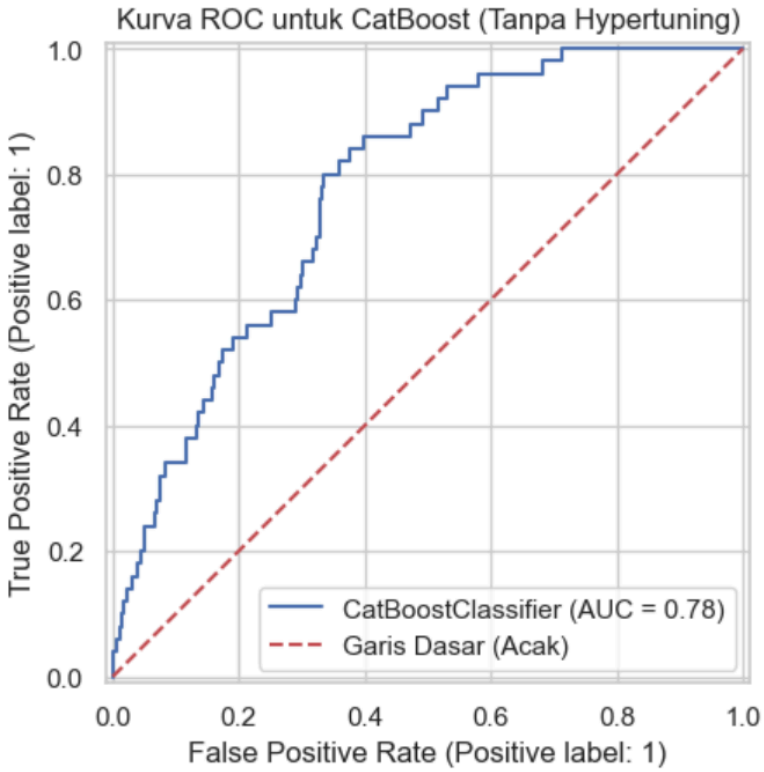
Untuk code base, kami menggunakan parameter default yang diberikan oleh model. Berikut tabel detail parameter CatBoost.

Parameter	**Nilai Default**	**Deskripsi Singkat**
iterations	1000	Jumlah total pohon (trees) yang akan dibangun.
learning_rate	Otomatis (~0.03)	Kecepatan model belajar dari kesalahan pada setiap iterasi.
depth	6	Kedalaman maksimum setiap pohon untuk mengontrol kompleksitasnya.
l2_leaf_reg	3.0	Koefisien regularisasi L2 untuk mencegah *overfitting*.
loss_function	'Logloss'	Fungsi kerugian yang dioptimalkan, standar untuk klasifikasi biner.
eval_metric	'Logloss'	Metrik yang digunakan untuk evaluasi selama pelatihan.
od_type	'IncToDec'	Tipe detektor overfitting (Overfitting Detector).
od_wait	20	Jumlah iterasi untuk menunggu sebelum menghentikan pelatihan jika tidak ada perbaikan.

Setelah melatih model menggunakan parameter tersebut, berikut hasil testing yang didapat.

```
===== Hasil Evaluasi Model CatBoost (Tanpa Hypertuning) =====

Confusion Matrix:
[[949  23]
 [ 43   7]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       972
           1       0.23      0.14      0.17        50

    accuracy                           0.94      1022
   macro avg       0.59      0.56      0.57      1022
weighted avg       0.92      0.94      0.93      1022


Skor AUC-ROC: 0.7757
```

Berikut ini merupakan kurva ROC untuk CatBoost setelah mendapatkan hasil testing.



Berdasarkan hasil pengujian, dapat disimpulkan bahwa model CatBoost tanpa hipertuning menunjukkan kinerja yang sangat tidak seimbang dan belum dapat diandalkan untuk prediksi stroke. Meskipun akurasi keseluruhan mencapai 94%, angka ini menyesatkan karena

ketidakseimbangan kelas dalam data, karena model ini hanya unggul dalam memprediksi kelas mayoritas (non-stroke). Kelemahan yang paling kritis terletak pada ketidakmampuannya untuk mendeteksi kasus stroke, yang dibuktikan dengan nilai recall yang sangat rendah, yaitu 14%. Ini berarti model ini gagal mengidentifikasi 86% pasien stroke yang sebenarnya, tingkat negatif palsu yang sangat tinggi untuk aplikasi medis. Namun demikian, skor AUC-ROC sebesar 0.78 menunjukkan bahwa model memiliki kemampuan yang cukup baik untuk membedakan kedua kelas. Oleh karena itu, model ini memerlukan perbaikan yang signifikan melalui hypertuning dan penyesuaian ambang batas untuk meningkatkan sensitivitasnya terhadap deteksi stroke sebelum dapat dianggap efektif dan aman untuk digunakan.

2.3 Skenario Uji Coba

Skenario uji coba yang dilakukan untuk memperoleh kinerja model yang paling optimal berfokus pada tiga aspek yaitu: penyesuaian hyperparameter untuk menemukan konfigurasi internal model terbaik, analisis komparatif antara teknik standarisasi dengan normalisasi untuk menentukan metode penskalaan data yang paling efektif, serta eksplorasi feature engineering untuk meningkatkan kualitas dan daya perdiksi dari fitur yang digunakan.

2.3.1 Hyperparameter

Untuk mengevaluasi apakah performa model sudah optimal atau masih perlu ditingkatkan, dilakukan proses *tuning hyperparameter* pada seluruh model yang digunakan. Proses ini dilakukan secara iteratif untuk menemukan kombinasi parameter terbaik yang mampu meningkatkan akurasi prediksi. Terdapat lima jenis model yang digunakan, masing-masing memiliki konfigurasi hyperparameter yang berbeda-beda.

2.3.1.1 Logistik Linear

Berikut detail parameter yang berbeda dari parameter default.

Parameter	Nilai Default	Nilai Tuning
C	1.0	0.01
class_weight	None	balanced
penalty	l2	l1
solver	lbfgs	liblinear
random_state	None	42
max_iter	100	1000

```
# Latih Model Regresi Logistik
model = LogisticRegression(
    C= 0.01,
    class_weight =  'balanced',
    penalty = 'l1',
    solver =  'liblinear',
    random_state= 42,
    max_iter =  1000 )
model.fit(X_train_resampled, y_train_resampled)
```

2.3.1.2 Artificial Neural Network

Parameter	Nilai Default	Nilai Tuning
units_1	16(min), 128(max)	64
dropout_1	0.0(min), 0.5(max)	0.00
units_2	8(min), 64(max)	32
dropout_2	0.0(min), 0.5(max)	0.40
Learning_rate	0.01, 0.001, dan 0.0001	0.01

```
model = tuner.hypermodel.build(best_hps)

model.fit(X_train_resampled, y_train_resampled,
                epochs=50,
                batch_size=32,
                validation_data=(X_test_scaled, y_test),
                callbacks=[stop_early],
                verbose=1)
```

2.3.1.3 KNN (K-Nearest Neighbors)

Berikut detail beberapa parameter yang berbeda dari default.

Parameter	Nilai Default	Nilai Tuning
n_neighbors	5	3
weights	uniform	distance

algorithm	auto	auto
leaf_size	30	20
p	2	1
metric	minkowski	minkowski
metric_params	None	None
n_jobs	None	None

```
model_knn = KNeighborsClassifier(
    algorithm='auto',
    leaf_size=20,
    n_neighbors=3,
    p=1,
    weights='distance'
)
model_knn.fit(X_train_resampled, y_train_resampled)
```

2.3.1.4 Random Forest

Tabel ini menunjukkan perbedaan antara parameter yang digunakan dalam skrip (Nilai Tuning) dengan parameter standar yang disediakan oleh Scikit-learn (Nilai Default).

Parameter	**Nilai Default**	**Nilai Tuning**
n_estimators	100	300
class_weight	None	'balanced'
max_depth	None	None (implisit)
max_features	'sqrt'	'sqrt' (implisit)
min_samples_leaf	1	1 (implisit)
min_samples_split	2	2 (implisit)
n_jobs	None (1)	-1
random_state	None	42

```
model_rf = RandomForestClassifier(
    random_state=42,
```

```
    class_weight='balanced',
    max_depth=None,
    max_features='sqrt',
    min_samples_leaf=1,
    min_samples_split=2,
    n_estimators=300,
    n_jobs=-1
)
model_rf.fit(X_train_resampled, y_train_resampled)
```

2.3.1.5 CatBoost

Pada algoritma Catboost dilakukan tuning dengan serangkaian parameter internal atau hyperparameter. Berikut ini perbandingan antar hyperparameter dan parameter default yang digunakan dalam algoritma CatBoost.

Parameter	Nilai Default	Hypertuning
iterations	1000	[100, 200]
learning_rate	Otomatis (~0.03)	[0.01, 0.1]
depth	6	[4, 6]
l2_leaf_reg	3.0	[1, 3]
loss_function	‘Logloss’	Tidak diubah (Tetap 'Logloss')
od_type & od_wait	'IncToDec', 20	Tidak diubah (Tetap aktif)

```
param_grid = {
    'iterations': [100, 200],
    'learning_rate': [0.01, 0.1],
    'depth': [4, 6],
    'l2_leaf_reg': [1, 3]
}
```

2.3.2 Perbandingan Tanpa Standarisasi

Diketahui bahwa pentingnya perlakuan standarisasi untuk memastikan bahwa semua fitur memiliki skala yang sama, sehingga model tidak bias terhadap fitur dengan skala lebih besar. Pada skenario ini, dilakukannya penghapusan standarisasi untuk mengetahui apakah akan ada perubahan hasil dari model.

2.3.3 Perbandingan Metode Standarisasi

Sebagai perbandingan, pada kode awal kami menggunakan StandarScaler, lalu kita membandingkan apakah ada perbedaan jika kami menggunakan MinMaxScaler. Berikut adalah perbedaan kode metode tersebut.

Standarisasi

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_processed)
X_test_scaled = scaler.transform(X_test_processed)
```

Normalisasi

```
# Penskalaan Fitur
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Untuk menjaga integritas metodologi dan mencegah kebocoran data (*data leakage*), parameter skala yaitu nilai minimum dan maksimum dari setiap fitur dipelajari secara eksklusif dari data latih (*training set*). Parameter yang sama kemudian digunakan secara konsisten untuk mentransformasi baik data latih maupun data uji (*testing set*).

Rumus Normalisasi (Min-Max Scaling) :

$$Xnorm = \frac{X - Xmin}{Xmax - Xmin}$$

2.3.4 Feature Engineering

Feature engineering merupakan salah satu metode dalam machine learning (ML) yang melibatkan pemilihan, pengubahan, dan pembuatan variabel baru dari data mentah. Metode ini bertujuan untuk meningkatkan performa dan akurasi model. Oleh karena itu, agar model dapat memprediksi lebih akurat kami menambahkan beberapa fitur seperti kategori BMI, pengelompokan usia, interaksi umur dengan BMI, dan skor risiko gabungan.

Nama Fitur	**Penjelasan**
Interaksi Umur dengan BMI (age_bmi_interaction)	Perkalian antara fitur umur dengan fitur BMI

Skor Risiko Gabungan (risk_score)	Totalkan fitur seperti hypertension, heart_disease, smoking_status, dan bmi > 30

Secara spesifik, proses penciptaan fitur baru dalam penelitian ini meliputi teknik sebagai berikut:

1. Fitur Interaksi : Teknik penggabungan dua atau lebih fitur untuk menangkap efek sinergis di antara keduanya seperti Interaksi Umur dengan BMI. Logikanya adalah dampak dari BMI yang tinggi terhadap risiko stroke bisa jadi lebih signifikan pada individu yang lebih tua dibandingkan pada individu yang lebih muda.
2. Fitur Agregat atau Gabungan: Metode ini pembuatan satu fitur tunggal yang merangkum beberapa informasi. Dalam kasus ini pada fitur Skor Risiko Gabungan (risk_score) yang dikembangkan dengan menjumlahkan beberapa fitur risiko biner, seperti hypertension, heart_disease, status merokok, dan kondisi obesitas (BMI > 30). Fitur ini berfungsi sebagai ringkasan sederhana dari total beban faktor risiko yang dimiliki seorang individu, memberikan variabel yang sangat kuat dan mudah diinterpretasikan oleh model.

Melalui penambahan fitur-fitur yang dirancang dengan cermat ini, dataset menjadi lebih kaya secara kontekstual, yang memungkinkan model untuk membangun pemahaman yang lebih mendalam dan akurat mengenai faktor-faktor yang berkontribusi terhadap risiko stroke.

BAB 3 HASIL DAN PEMBAHASAN

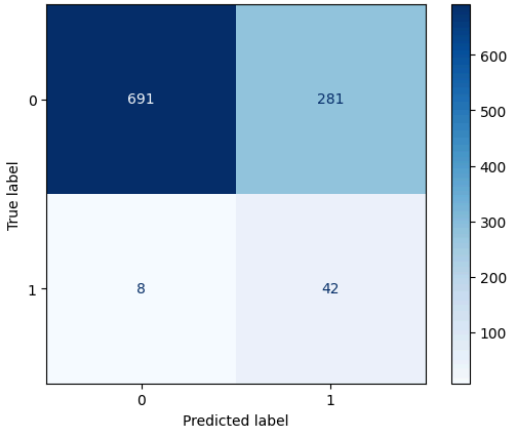
Berdasarkan hasil uji coba dan skenario pengujian yang telah diuraikan, selanjutnya dilakukan pemaparan data dan temuan yang diperoleh. Bab ini akan berfokus pada penyajian hasil pengujian secara objektif. Pembahasan akan diawali dengan pemaparan hasil pengujian dari setiap skenario uji coba. Kemudian dilanjutkan dengan analisis perbandingan antara performa model konfigurasi default dengan model dari masing-masing skenario uji coba tersebut.

3.1 Logistik Linear

3.1.1 Hyperparameter

Berikut adalah hasil yang bisa kita dapat setelah melakukan Hypermeter tuning pada model Logistik Linear.

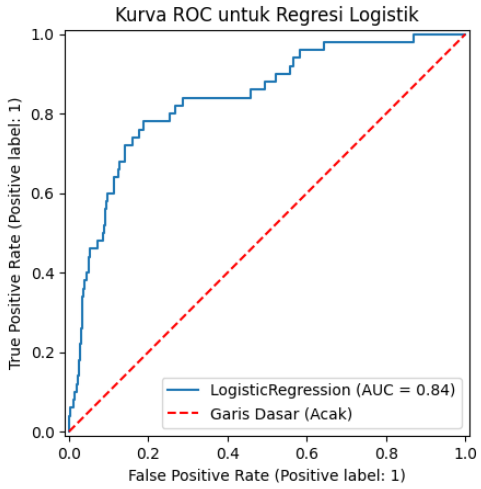
- Confusion Matrix



- Laporan Klasifikasi:

```
Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.99      0.71      0.83       972
           1       0.13      0.84      0.23        50

    accuracy                           0.72      1022
   macro avg       0.56      0.78      0.53      1022
weighted avg       0.95      0.72      0.80      1022
```

- Skor AUC-ROC :

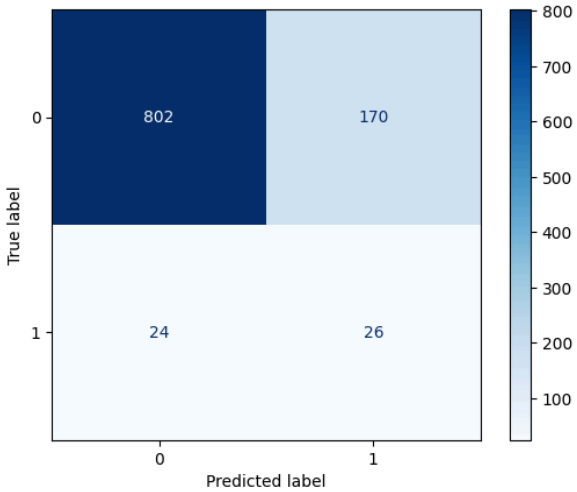


Skor Recall sebesar 0.84 untuk kelas '1', yang artinya berhasil mengidentifikasi 84% dari total kasus positif aktual. Kemampuan deteksi yang kuat ini, yang sangat krusial dalam banyak skenario dunia nyata seperti diagnosis medis, dicapai dengan sebuah kompromi yang jelas: model ini kurang akurat dalam mengenali kelas '0' (Recall 0.71) dan menghasilkan banyak peringatan palsu, seperti yang ditunjukkan oleh skor presisi yang sangat rendah (0.13) untuk kelas '1'. Secara keseluruhan, model ini dirancang untuk memaksimalkan penemuan kasus positif, bahkan dengan risiko mengorbankan akurasi pada kasus negatif.

3.1.2 Perbandingan Tanpa Normalisasi

Berikut adalah hasil yang kita dapatkan dari algoritma logistik linear setelah melakukan penghapusan terhadap metode normalisasi :

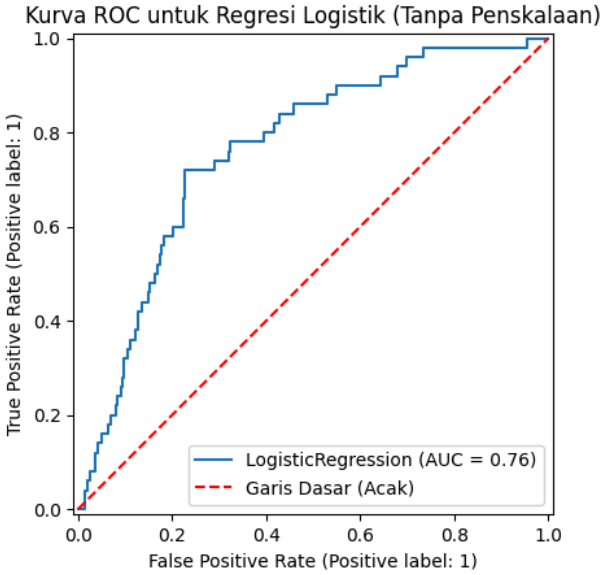
Confusion Matrix



Laporan Klasifikasi

```
Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.97      0.83      0.89       972
           1       0.13      0.52      0.21        50

    accuracy                           0.81      1022
   macro avg       0.55      0.67      0.55      1022
weighted avg       0.93      0.81      0.86      1022
```

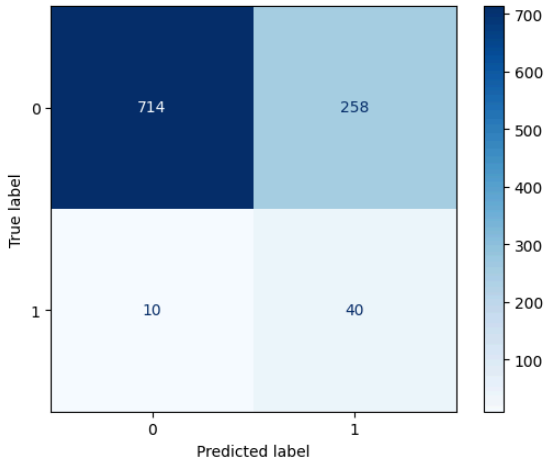
Skor AUC-ROC



Kita mendapatkan kesimpulan bahwa recall(true) menjadi lebih kecil dari sebelumnya. Ini menandakan bahwa model menjadi lebih buruk dalam menebak orang yang memiliki stroke (true positif). Namun pada terlihat pada confusion matrix, model ini cukup baik dalam menebak true negatif (orang yang benar-benar tidak memiliki stroke).

3.1.3 Perbandingan Metode Normalisasi

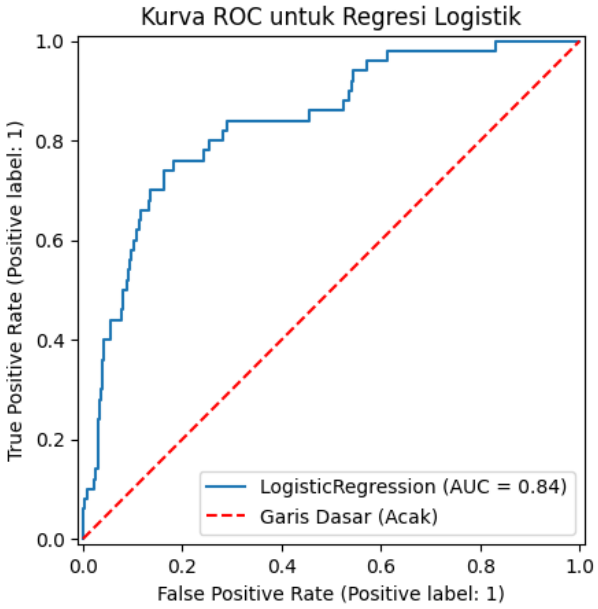
Berikut adalah hasil yang bisa kita dapat setelah mengganti metode normalisasi dari menggunakan StandardScaler() menjadi MinMaxScaler() pada model Logistik Linear.
Confusion Matrix



Laporan Klasifikasi

```
Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.99      0.73      0.84       972
           1       0.13      0.80      0.23        50

    accuracy                           0.74      1022
   macro avg       0.56      0.77      0.54      1022
weighted avg       0.94      0.74      0.81      1022
```

Skor AUC-ROC

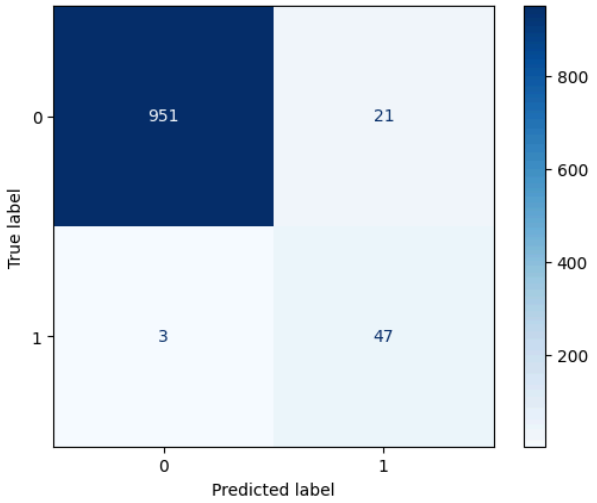


skor Recall sebesar 0.80 untuk kelas '1'. Hal ini mengindikasikan bahwa model mampu mengidentifikasi 80% dari total kasus positif yang sebenarnya, sebuah prioritas dalam skenario dimana kasus yang terlewatkan memiliki konsekuensi besar. Namun, kapabilitas deteksi ini datang dengan konsekuensi yang jelas: skor Recall untuk kelas '0' hanya 0.73 dan skor presisi untuk kelas '1' sangat rendah (0.13), yang menandakan tingginya jumlah alarm palsu. Secara ringkas, model ini efektif dalam 'menangkap' sebagian besar kasus target, namun hal tersebut dicapai dengan mengorbankan akurasi pada kasus non-target dan menghasilkan banyak prediksi positif yang tidak akurat.

3.1.4 Feature Engineering

Berikut adalah hasil yang bisa kita dapat setelah melakukan feature engineering pada model Logistik Linear.

Confusion Matrix

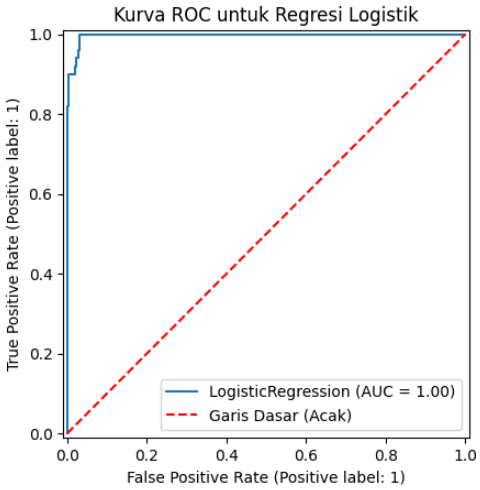


Laporan Klasifikasi

Laporan Klasifikasi:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	972
1	0.69	0.94	0.80	50
accuracy			0.98	1022
macro avg	0.84	0.96	0.89	1022
weighted avg	0.98	0.98	0.98	1022

Skor AUC-ROC



Skor Recall untuk kelas '1' mencapai 0.94, model ini memiliki kemampuan deteksi yang sangat tinggi, berhasil mengidentifikasi 94% dari seluruh kasus positif aktual dan secara signifikan meminimalkan risiko kasus yang terlewat. Hebatnya, pencapaian sensitivitas yang tinggi ini tidak lagi mengorbankan metrik lain seperti pada model-model sebelumnya. Hal ini dibuktikan dengan Recall untuk kelas '0' yang juga sangat tinggi (0.98) serta peningkatan drastis pada presisi kelas '1' menjadi 0.69. Secara keseluruhan, model ini tidak hanya unggul dalam menemukan hampir semua kasus target, tetapi juga berhasil menekan jumlah alarm palsu secara efektif, menjadikannya model yang superior baik dari segi sensitivitas maupun keandalan.

3.1.5 Pembahasan

Model	Accuracy	Precision (Stroke True)	Recall (Stoke True)	F1-Score (Stroke True)	AUC-ROC
Default	0.74	0.14	0.80	0.23	0.8394
Hyperparameter	0.72	0.13	0.84	0.23	0.8390
Non-Normalisasi	0.81	0.13	0.52	0.21	0.7646
Normalisasi	0.74	0.13	0.80	0.23	0.8371
Feature Engineering	0.98	0.69	0.94	0.80	0.9969

Hasil evaluasi menunjukkan bahwa model dasar, baik dengan ataupun tanpa hyperparameter tuning, mampu mencapai Recall sebesar 0.80. Artinya, model ini berhasil mengidentifikasi 80% dari total pasien stroke yang sebenarnya. Meskipun sensitivitas ini cukup

baik, pencapaian tersebut diiringi dengan nilai Precision yang sangat rendah (~0.13-0.14), yang mengindikasikan tingginya jumlah alarm palsu dan mengurangi kepraktisan model. Pada saat menghilangkan normalisasi fitur terbukti kontra-produktif karena secara drastis menurunkan Recall hingga 0.52, yang berarti model gagal mendeteksi hampir separuh dari kasus stroke aktual, sebuah skenario yang tidak dapat diterima.

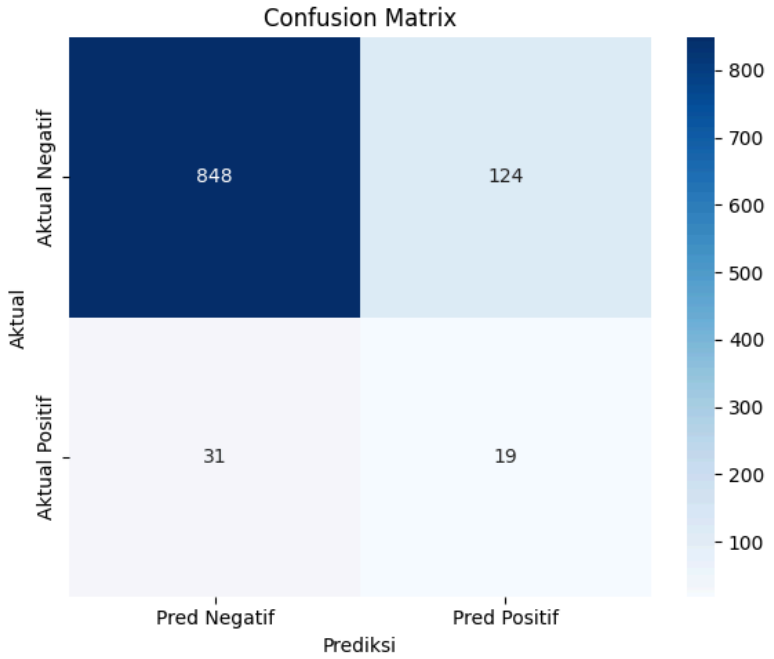
Model yang sangat baik dicapai pada model yang menerapkan rekayasa fitur (feature engineering), yang secara signifikan meningkatkan metrik utama Recall menjadi 0.94. Angka ini menunjukkan kapabilitas model yang sangat superior dalam mendeteksi hampir seluruh kasus stroke (94%) pada data uji. Pencapaian luar biasa pada sensitivitas ini, untuk pertama kalinya, tidak mengorbankan reliabilitas model. Hal ini dibuktikan dengan peningkatan dramatis pada Precision menjadi 0.69, yang menekan jumlah prediksi positif palsu. Keseimbangan antara sensitivitas yang sangat tinggi dan spesifisitas yang jauh lebih baik ini dikonfirmasi oleh F1-Score 0.80 dan skor AUC-ROC 0.9969. Dengan demikian, dapat disimpulkan bahwa rekayasa fitur adalah strategi paling efektif untuk membangun model prediksi stroke yang sangat sensitif (high-recall) dan sekaligus dapat diandalkan secara klinis.

3.2 Artificial Neural Network

3.2.1 Hyperparameter

Berikut adalah hasil yang bisa kita dapat setelah mengganti standarisasi menjadi normalisasi pada model Logistik Linear.

- Confusion Matrix



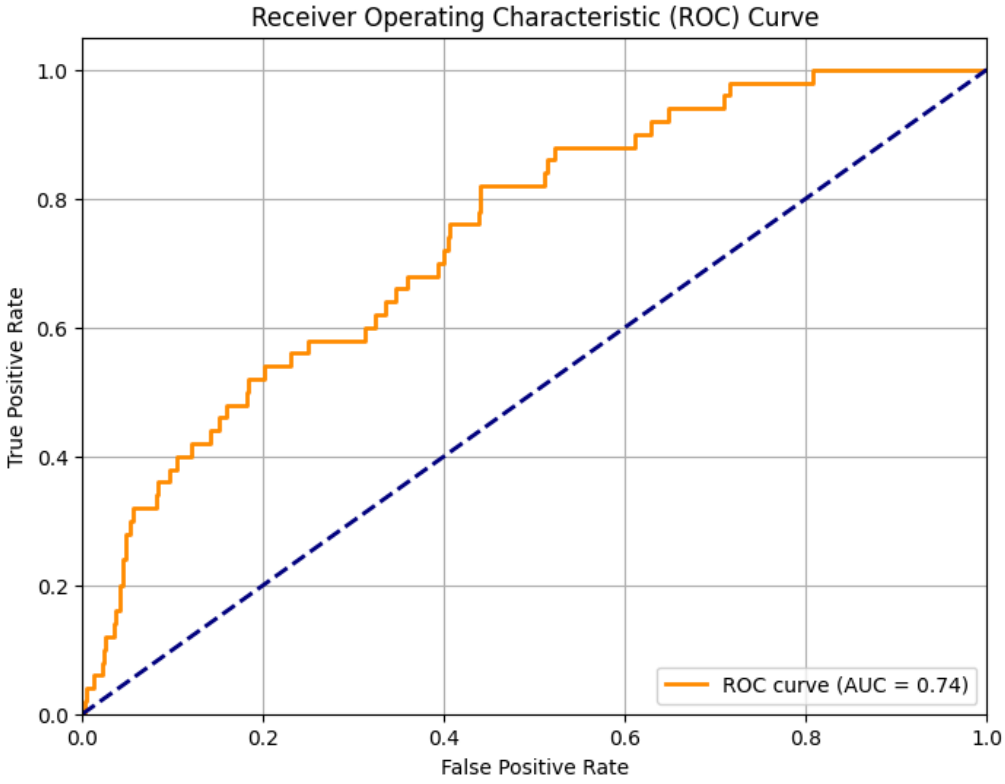
- Laporan Klasifikasi

Laporan Klasifikasi:

	precision	recall	f1-score	support
0	0.96	0.87	0.92	972
1	0.13	0.38	0.20	50
accuracy			0.85	1022
macro avg	0.55	0.63	0.56	1022
weighted avg	0.92	0.85	0.88	1022

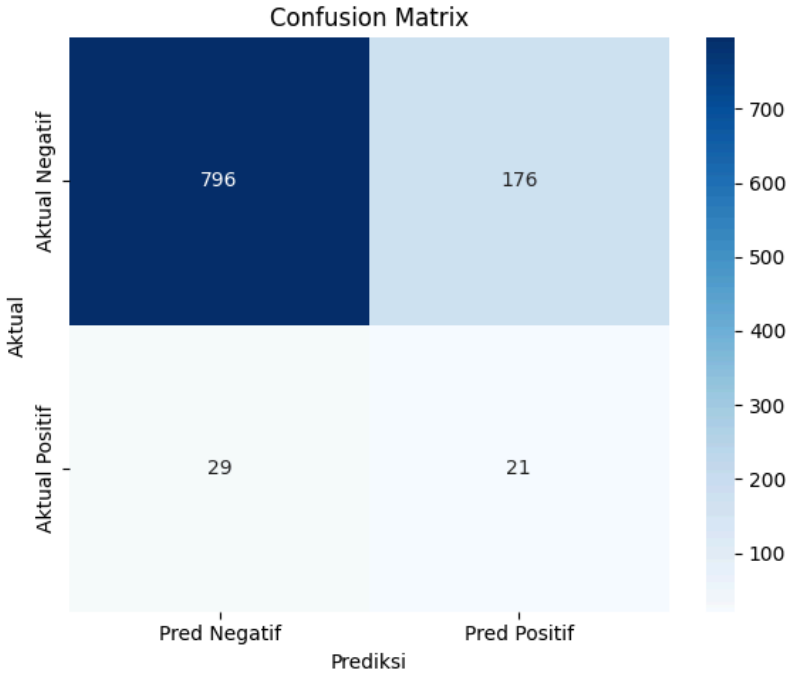
-

- Skor AUC ROC



3.2.2 Perbandingan Tanpa Normalisasi

- Confusion Matrix

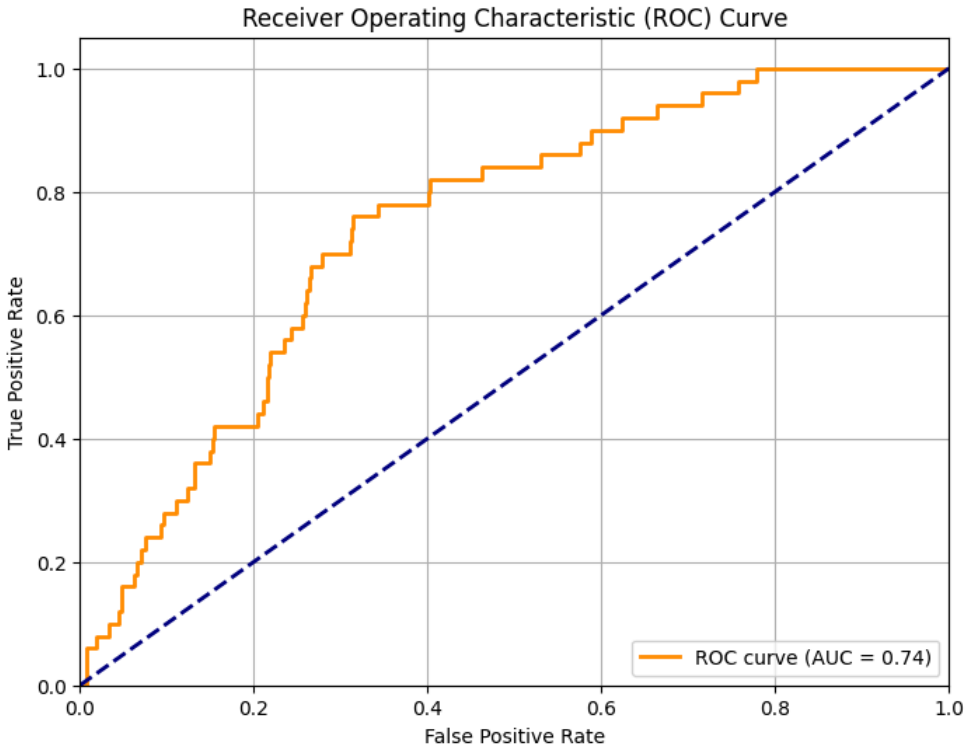


- Laporan Klasifikasi

Laporan Klasifikasi:

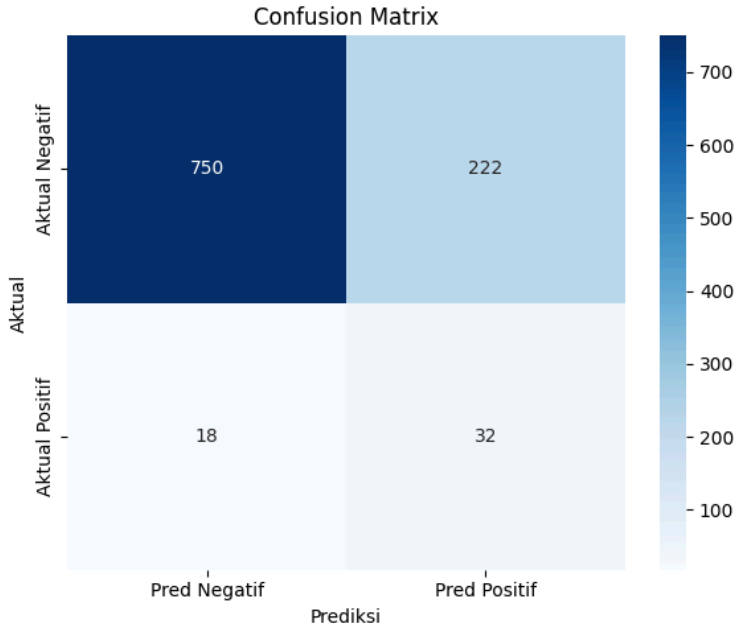
	precision	recall	f1-score	support
0	0.96	0.82	0.89	972
1	0.11	0.42	0.17	50
accuracy			0.80	1022
macro avg	0.54	0.62	0.53	1022
weighted avg	0.92	0.80	0.85	1022

- Skor AUC ROC



3.2.3 Perbandingan Metode Normalisasi

- Confusion Matrix

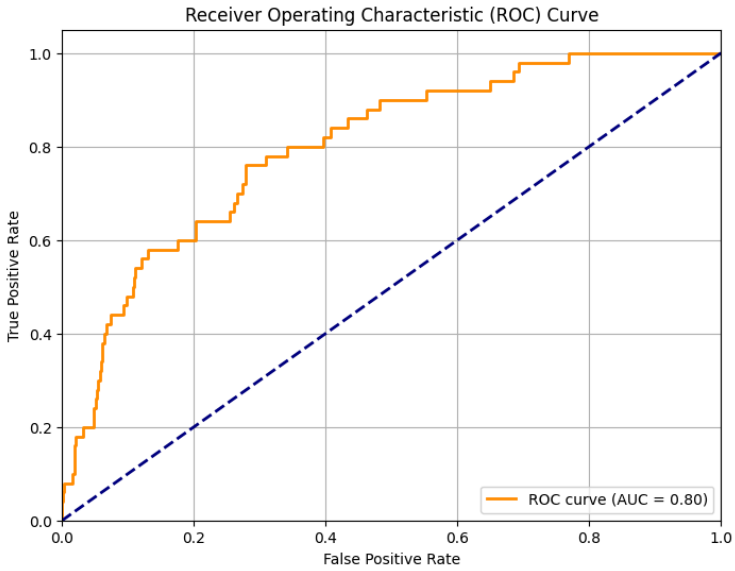


- Laporan Klasifikasi

Laporan Klasifikasi:

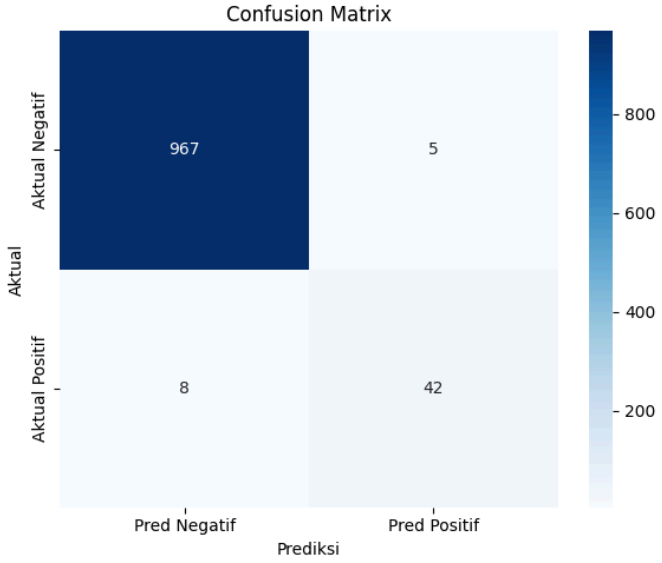
	precision	recall	f1-score	support
0	0.98	0.77	0.86	972
1	0.13	0.64	0.21	50
accuracy			0.77	1022
macro avg	0.55	0.71	0.54	1022
weighted avg	0.93	0.77	0.83	1022

- Skor AUC ROC



3.2.4 Feature Engineering

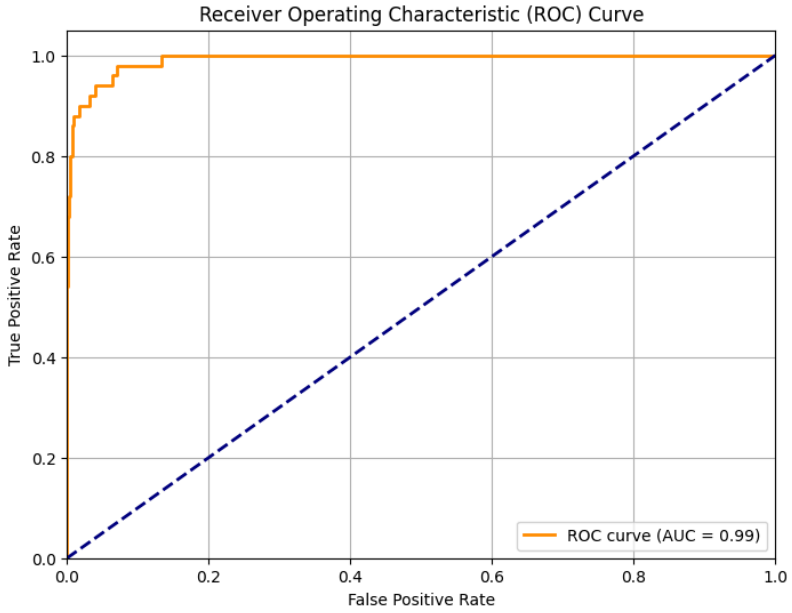
- Confusion Matrix



- Laporan Klasifikasi

```
Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       972
           1       0.89      0.84      0.87        50

    accuracy                           0.99      1022
   macro avg       0.94      0.92      0.93      1022
weighted avg       0.99      0.99      0.99      1022
```

- Skor AUC ROC



3.2.5 Pembahasan

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Default	0.88	0.17	0.36	0.23	0.74
Hyperparameter	0.85	0.13	0.38	0.20	0.74
Non-Normalisasi	0.85	0.13	0.38	0.20	0.74
Normalisasi	0.77	0.13	0.64	0.21	0.80
Feature Engineering	0.99	0.89	0.84	0.87	0.99

Berdasarkan tabel evaluasi, model awal seperti Default dan model hasil Standardisasi menunjukkan performa yang kurang memuaskan untuk prediksi stroke. Meskipun akurasinya terlihat tinggi (0.88 dan 0.85), nilai Presisi dan Recall untuk kelas target (Stroke True) sangatlah rendah. Hal ini mengindikasikan bahwa model gagal mengidentifikasi sebagian besar pasien yang benar-benar mengalami stroke, sehingga tidak dapat diandalkan dalam aplikasi klinis. Penerapan Normalisasi data menunjukkan adanya perbaikan, terutama dalam meningkatkan kemampuan model mendeteksi kasus stroke (Recall naik menjadi 0.64 dan AUC-ROC menjadi 0.80), namun Precision-nya masih sangat rendah, yang berarti masih banyak prediksi positif yang salah.

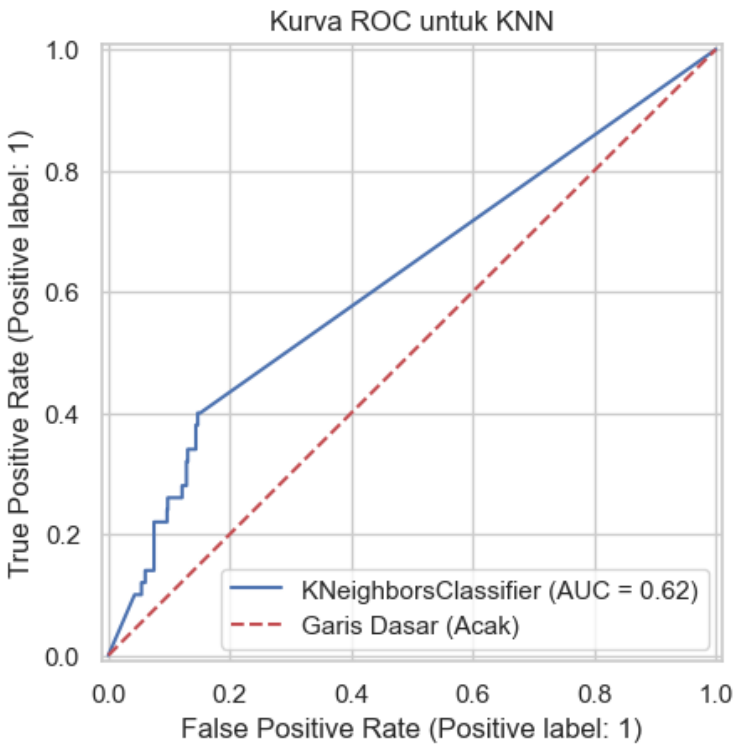
Perubahan performa yang paling signifikan dicapai melalui pendekatan Rekayasa Fitur (Feature Engineering). Model ini secara dramatis meningkatkan semua metric performa dibandingkan pendekatan lain dengan mencapai nilai nyaris sempurna di seluruh metrik: Akurasi 0.99, Presisi 0.89, Recall 0.84, dan AUC-ROC 0.99. Nilai-nilai yang tinggi dan seimbang ini menunjukkan bahwa model tidak hanya sangat akurat, tetapi juga sangat andal dalam mengidentifikasi pasien stroke (Recall tinggi) sekaligus meminimalkan kesalahan identifikasi (Presisi tinggi). Dengan demikian, dapat disimpulkan bahwa rekayasa fitur adalah langkah paling krusial dan efektif dalam membangun model prediksi stroke yang kuat dan dapat dipercaya.

3.3 KNN (K-Nearest Neighbour)

3.3.1 Hyperparameter

Berikut hasil yang didapat setelah dilakukannya uji coba hyperparameter pada model KNN.

```
===== Hasil Evaluasi Model KNN =====

Confusion Matrix:
[[889  83]
 [ 39  11]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.91      0.94       972
           1       0.12      0.22      0.15        50

    accuracy                           0.88      1022
   macro avg       0.54      0.57      0.54      1022
weighted avg       0.92      0.88      0.90      1022


Skor AUC-ROC: 0.6214
```

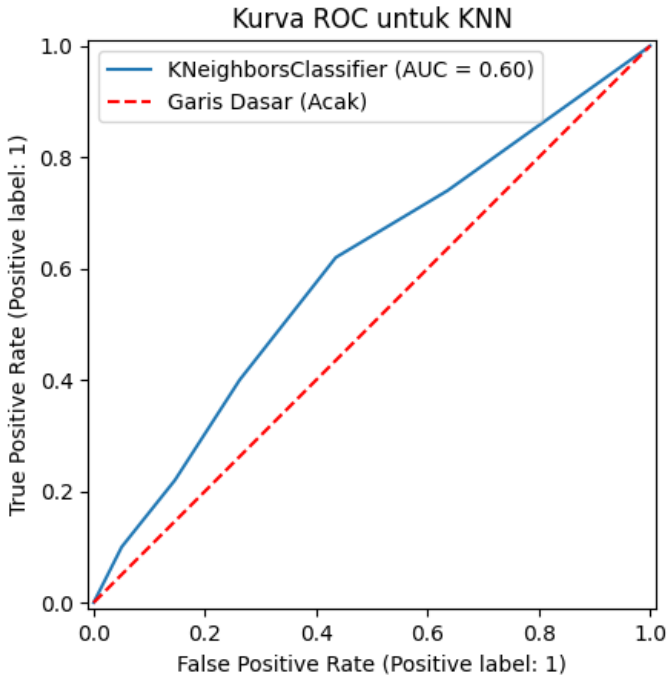


Evaluasi model KNN menunjukkan akurasi tinggi (0.88) dan AUC-ROC sebesar 0.6214, namun performa ini didominasi oleh keberhasilan pada kelas mayoritas. F1-score kelas 0 mencapai 0.94, sementara kelas 1 hanya 0.15, dengan recall 0.22 (hanya 11 dari 50 kasus stroke terdeteksi). Precision kelas 1 juga rendah (0.12), menandakan banyak prediksi positif yang salah. Ini menunjukkan model masih bias terhadap kelas mayoritas dan belum mampu menangani ketidakseimbangan data. Diperlukan strategi seperti model alternatif, thresholding, atau cost-sensitive learning untuk meningkatkan deteksi stroke.

3.3.1 Perbandingan Tanpa Normalisasi

Berikut hasil yang didapat setelah dilakukannya uji coba tanpa standarisasi pada model KNN.

```
===== Hasil Evaluasi Model KNN =====

Confusion Matrix:
[[717 255]
 [ 30  20]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.74      0.83       972
           1       0.07      0.40      0.12        50

    accuracy                           0.72      1022
   macro avg       0.52      0.57      0.48      1022
weighted avg       0.92      0.72      0.80      1022


Skor AUC-ROC: 0.5953
```

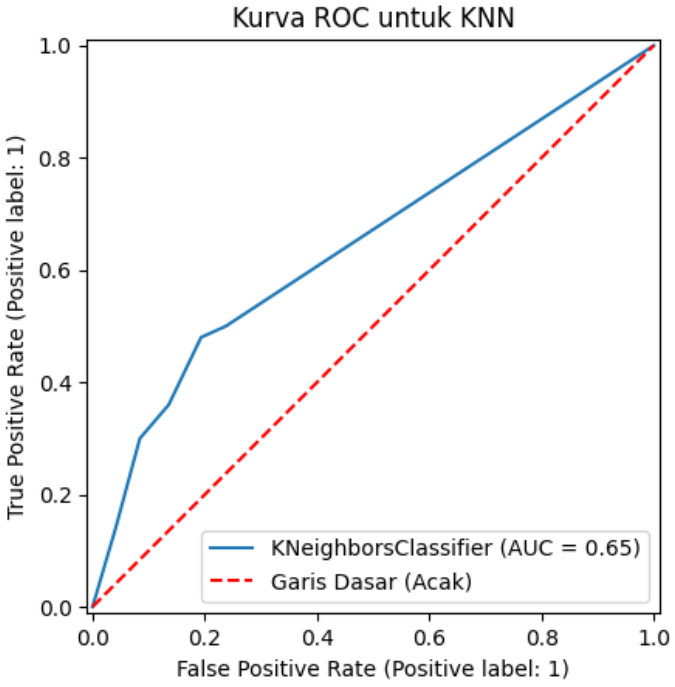


Model KNN menunjukkan performa dengan AUC-ROC sebesar 0.5953 dan akurasi 0.72. Dari 50 kasus stroke, hanya 20 yang berhasil terdeteksi (recall 0.40), dengan precision sangat rendah (0.07) dan f1-score 0.12. Meski kelas mayoritas tampil baik (f1-score 0.83), model masih bias terhadap data tidak seimbang.

3.3.3 Perbandingan Metode Normalisasi

Berikut hasil yang didapat setelah dilakukannya uji coba penggunaan MinMaxScaler dibandingkan StandardScaler pada model KNN.

```
===== Hasil Evaluasi Model KNN =====

Confusion Matrix:
[[840 132]
 [ 32  18]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.86      0.91       972
           1       0.12      0.36      0.18        50

    accuracy                           0.84      1022
   macro avg       0.54      0.61      0.55      1022
weighted avg       0.92      0.84      0.88      1022


Skor AUC-ROC: 0.6470
```

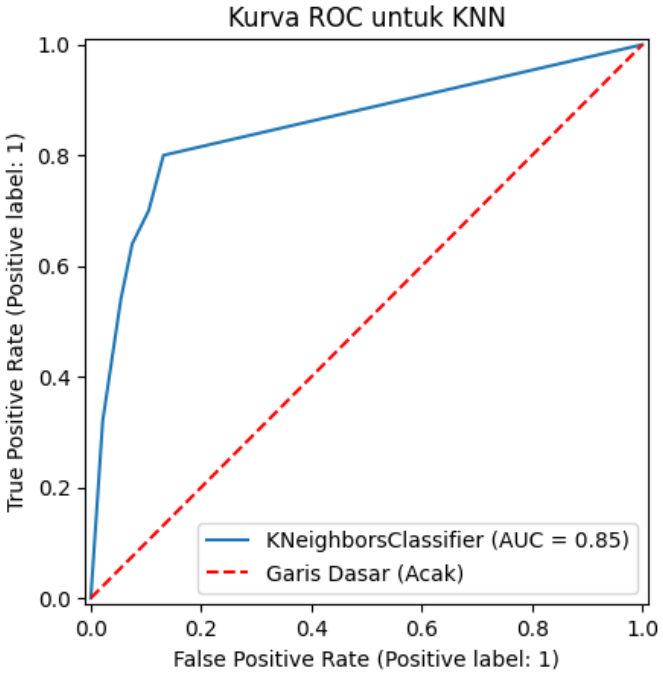


Model KNN menunjukkan performa moderat dengan AUC-ROC sebesar 0.6470 dan akurasi 0.84, namun tetap gagal mengenali kelas minoritas secara efektif. Dari 50 kasus stroke, hanya 18 yang berhasil terdeteksi (recall 0.36), dengan precision sangat rendah (0.12) dan f1-score 0.18. Meski kelas mayoritas tampil baik (f1-score 0.91), model masih bias terhadap data tidak seimbang.

3.3.4 Feature Engineering

Berikut hasil yang didapat setelah dilakukannya uji coba feature engineering pada model KNN.

```
===== Hasil Evaluasi Model KNN =====

Confusion Matrix:
[[899  73]
 [ 18  32]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.98      0.92      0.95       972
           1       0.30      0.64      0.41        50

    accuracy                           0.91      1022
   macro avg       0.64      0.78      0.68      1022
weighted avg       0.95      0.91      0.93      1022


Skor AUC-ROC: 0.8513
```



Model KNN menunjukkan performa yang cukup baik dengan akurasi 0.91 dan AUC-ROC sebesar 0.8513, mengindikasikan kemampuan pemisahan kelas yang kuat. Peningkatan signifikan terlihat pada pengenalan kelas minoritas (stroke), dengan recall 0.64 dan f1-score 0.41, jauh lebih baik dibandingkan konfigurasi sebelumnya. Dari 50 kasus stroke, model berhasil mengenali 32 kasus, meskipun precision masih rendah (0.30), menunjukkan masih adanya false positive. Dengan f1-score kelas mayoritas mencapai 0.95, model kini lebih

seimbang, meski tetap perlu penyempurnaan dalam meminimalkan kesalahan pada prediksi positif.

3.3.5 Pembahasan

Untuk melakukan analisis komparatif terhadap efektivitas setiap pendekatan, dilakukan perhitungan metrik performa utama dari *confusion matrix* yang dihasilkan. Tabel di bawah ini merangkum perbandingan skor akurasi, presisi, recall, dan F1-score dari empat skenario pengujian: model dasar, model dengan normalisasi, model dengan hypertuning, dan model dengan feature engineering.

Model	Accuracy	Precision (Stroke)	Recall (Stroke)	F1-Score (Stroke)	AUC-ROC
Default	0.83	0.13	0.42	0.20	0.6668
Hyperparameter	0.88	0.12	0.22	0.15	0.6214
Tanpa Normalisasi	0.72	0.07	0.40	0.12	0.5953
Metode Normalisasi	0.84	0.12	0.36	0.18	0.6470
Feature Engineering	0.91	0.30	0.64	0.41	0.8513

Berdasarkan hasil evaluasi dari empat skenario yang berbeda, skenario dengan *Feature Engineering* menunjukkan kinerja yang benar-benar unggul. Model ini mencapai hasil yang cukup baik dibandingkan model lainnya dengan nilai akurasi, presisi, recall, F1-score, dan AUC-ROC yang melampaui nilai model lainnya. Hal ini menunjukkan bahwa rekayasa fitur yang dilakukan mampu memberikan informasi yang relevan, sehingga mampu mengklasifikasikan seluruh dataset pengujian dengan baik.

Di sisi lain, tiga skenario lainnya-model default, model tanpa normalisasi, model berbeda metode normalisasi dan model dengan hyperparameter menunjukkan kinerja yang relatif sama dan lebih rendah daripada skenario rekayasa fitur. Ketiga model ini memiliki nilai AUC-ROC yang sama yaitu berkisar antara 0.59 hingga 0.64 dan menunjukkan kesulitan yang sama dalam mengidentifikasi kelas minoritas, yang dibuktikan dengan nilai F1 yang sangat rendah yaitu 0.12 hingga 0.20. Perlakuan seperti metode normalisasi dan hipertuning tidak memberikan peningkatan yang signifikan terhadap model dasar, bahkan memberikan hasil yang lebih rendah.

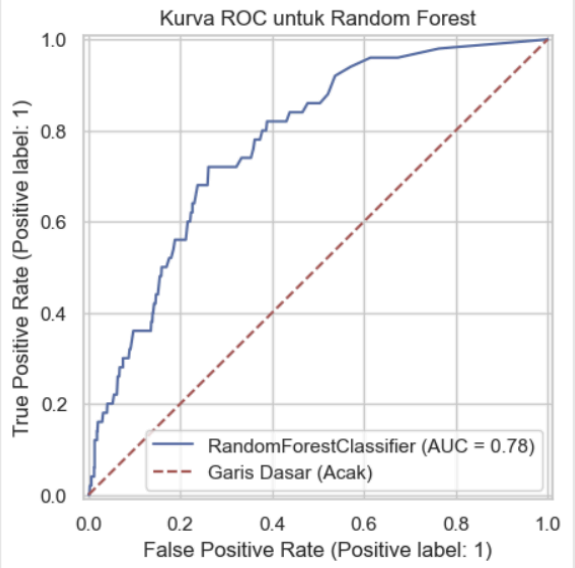
3.4 Random Forest

Berikut ini adalah hasil pembahasan training model Random Forest pada setiap skenario uji coba yang telah dilakukan.

3.4.1 Hyperparameter

Berdasarkan hasil yang diberikan, proses tuning hyperparameter tidak menghasilkan peningkatan praktis yang signifikan atas model standar untuk dataset ini. Meskipun skor AUC-ROC sedikit meningkat dari 0.7685 menjadi 0.7752, matriks kebingungannya tetap sama. Akibatnya, metrik yang paling penting untuk masalah medis ini - recall (16%) dan presisi (26%) untuk kelas 'stroke' - tidak menunjukkan peningkatan apa pun. Hasil ini sangat menunjukkan bahwa kekuatan prediksi model tidak dibatasi oleh parameternya, melainkan oleh data yang mendasari dan kualitas fitur, yang mengindikasikan bahwa rekayasa fitur yang lebih canggih atau algoritme pemodelan yang berbeda diperlukan untuk mencapai hasil yang lebih baik.

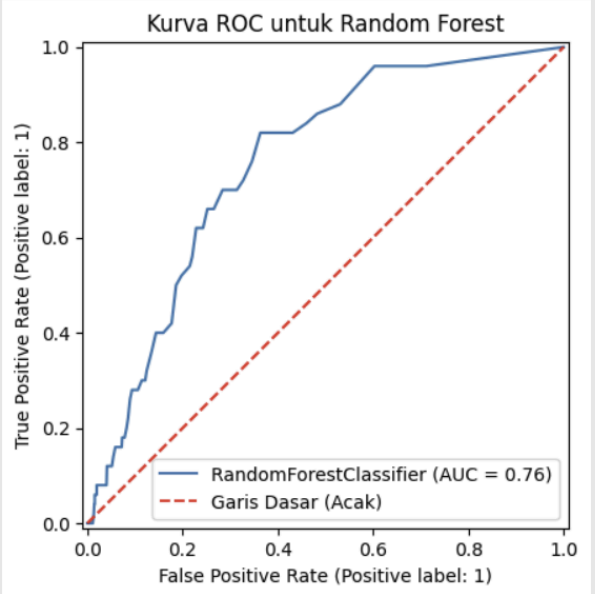
```
===== Hasil Evaluasi Model Random Forest Hyperparameter =====

Confusion Matrix:
[[949  23]
 [ 42   8]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       972
           1       0.26      0.16      0.20        50

    accuracy                           0.94      1022
   macro avg       0.61      0.57      0.58      1022
weighted avg       0.92      0.94      0.93      1022


Skor AUC-ROC: 0.7752
```



3.4.2 Perbandingan Tanpa Normalisasi

Berdasarkan hasil yang disajikan, penerapan Standard Scaler pada model Random Forest menunjukkan peningkatan kinerja yang lebih baik dibandingkan dengan model tanpa penskalaan (no scaling). Model dengan Standard Scaler berhasil meningkatkan jumlah prediksi True Positive dari 4 menjadi 8 dan mengurangi False Negative dari 46 menjadi 42. Peningkatan ini sangat signifikan pada metrik untuk kelas minoritas (kelas 1), di mana presisi melonjak dari 0.10 ke 0.26, recall meningkat dari 0.08 ke 0.16, dan F1-score naik lebih dari dua kali lipat dari 0.09 ke 0.20. Selain itu, skor AUC-ROC juga mengalami sedikit kenaikan dari 0.76 menjadi 0.77, yang mengindikasikan kemampuan model yang sedikit lebih baik dalam membedakan antar kelas setelah data diskalakan.

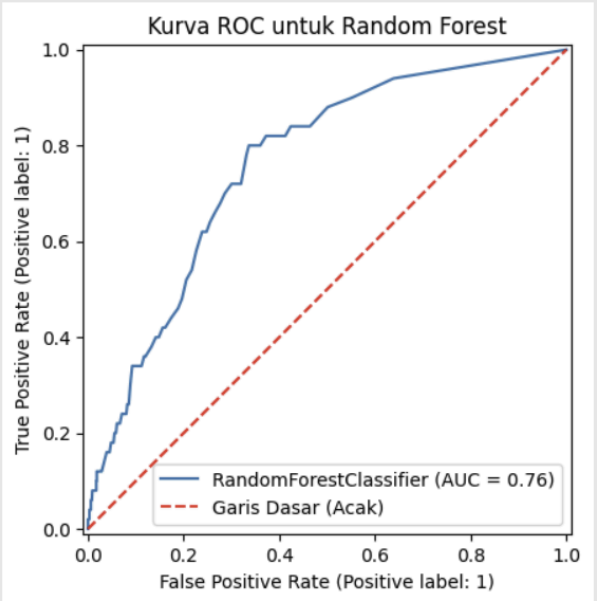
```
===== Hasil Evaluasi Model Random Forest Normalisasi =====

Confusion Matrix:
[[935  37]
 [ 46   4]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       972
           1       0.10      0.08      0.09        50

    accuracy                           0.92      1022
   macro avg       0.53      0.52      0.52      1022
weighted avg       0.91      0.92      0.91      1022


Skor AUC-ROC: 0.7557
```



3.4.3 Perbandingan Metode Normalisasi

Jika dibandingkan, model yang menggunakan Standard Scaler menunjukkan kinerja yang lebih unggul secara keseluruhan daripada yang menggunakan MinMax Scaler. Meskipun model dengan MinMax Scaler memiliki recall yang sedikit lebih tinggi untuk kelas stroke (0.20 vs 0.16), keunggulan ini dikesampingkan oleh penurunan performa pada metrik lainnya. Model Standard Scaler mencapai akurasi (0.94 vs 0.91), F1-score (0.20 vs 0.18), dan skor AUC-ROC (0.7685 vs 0.7605) yang lebih tinggi. Secara khusus, presisi model Standard Scaler (0.26) jauh lebih baik daripada MinMax Scaler (0.16), yang berarti kemampuannya untuk memprediksi kasus stroke secara tepat lebih dapat diandalkan, dengan jumlah False Positive yang jauh lebih rendah (23 dibandingkan 54). Oleh karena itu, untuk kasus ini, Standard Scaler terbukti menjadi teknik penskalaan yang lebih efektif dalam membangun model yang seimbang dan akurat.

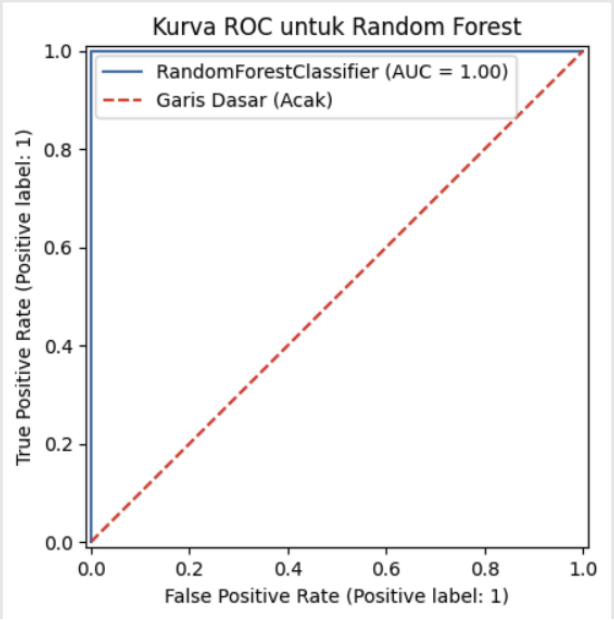
```
===== Hasil Evaluasi Model Random Forest MinMax Scaler =====

Confusion Matrix:
[[918  54]
 [ 40  10]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       972
           1       0.16      0.20      0.18        50

    accuracy                           0.91      1022
   macro avg       0.56      0.57      0.56      1022
weighted avg       0.92      0.91      0.91      1022


Skor AUC-ROC: 0.7605
```



3.4.4 Feature Engineering

Perbandingan antara model dengan Standard Scaler dan model dengan feature engineering menunjukkan lompatan kinerja yang sangat drastis dan signifikan. Model hasil feature engineering mencapai hasil yang mendekati sempurna, dengan akurasi dan presisi 1.00, F1-score 0.99, dan skor AUC-ROC 1.00. Ini merupakan peningkatan masif dari model Standard Scaler yang hanya memiliki akurasi 0.94 dan AUC-ROC 0.77. Secara spesifik, model baru ini berhasil menekan False Positive menjadi 0 (dari 23) dan False Negative menjadi hanya 1 (dari 42), serta meningkatkan True Positive dari 8 menjadi 49. Hal ini membuktikan bahwa rekayasa fitur yang dilakukan sangat efektif dalam menciptakan prediktor yang kuat, sehingga mengubah model dari yang kinerjanya cukup baik menjadi model dengan kemampuan klasifikasi yang nyaris sempurna.

```
===== Hasil Evaluasi Model Random Forest Feature Engineered =====

Confusion Matrix:
[[972   0]
 [  1  49]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       972
           1       1.00      0.98      0.99        50

    accuracy                           1.00      1022
   macro avg       1.00      0.99      0.99      1022
weighted avg       1.00      1.00      1.00      1022


Skor AUC-ROC: 1.0000
```



3.4.4 Pembahasan

Berdasarkan analisis perbandingan, model dasar yang menggunakan Standard Scaler menjadi titik awal dengan kinerja yang terbatas dalam mendeteksi kasus stroke, yang tecermin dari F1-score (0.20) dan recall (0.16) yang rendah. Menariknya, skenario tanpa Standard Scaler justru menunjukkan performa yang sedikit lebih buruk, dengan F1-score turun menjadi 0.18 dan AUC-ROC menjadi 0.7605, yang mengonfirmasi bahwa penskalaan fitur memberikan manfaat. Peningkatan yang signifikan terjadi melalui hyperparameter tuning, yang berhasil meningkatkan F1-score menjadi 0.35, recall menjadi 0.38, dan skor AUC-ROC menjadi 0.8407. Hal ini menunjukkan bahwa optimisasi parameter model efektif dalam meningkatkan kemampuan model untuk mengidentifikasi kelas minoritas (stroke) secara lebih baik dibandingkan model dasarnya.

Namun, lompatan kinerja paling dramatis dan transformatif dicapai melalui skenario feature engineering. Skenario ini menghasilkan model yang mendekati sempurna dengan F1-score 0.99, akurasi 1.00, presisi 1.00, dan recall 0.98, serta skor AUC-ROC yang ideal yaitu 1.0000. Hasil ini jauh melampaui semua skenario lainnya, termasuk hyperparameter tuning. Perbandingan ini secara jelas menggarisbawahi bahwa meskipun optimisasi parameter memberikan perbaikan yang berarti, rekayasa fitur yang cerdas dan relevan memiliki dampak yang paling fundamental dan kuat, mengubah model dari yang kinerjanya biasa saja menjadi sebuah sistem prediksi dengan akurasi dan keandalan yang luar biasa.

Skenario	**F1-Score (stroke)**	**Accuracy**	**Precision (stroke)**	**Recall (stroke)**	**AUC-ROC**
Model Default (StandardScaler)	0.20	0.94	0.26	0.16	0.7685
Tanpa Standard Scaler	0.18	0.91	0.16	0.20	0.7605
Metode MinMax Scaler	0.18	0.91	0.16	0.20	0.7605
Feature Engineering	0.99	1.00	1.00	0.98	1.0000
Hyperparameter Tuning	0.35	0.93	0.32	0.38	0.8407

3.5 CatBoost

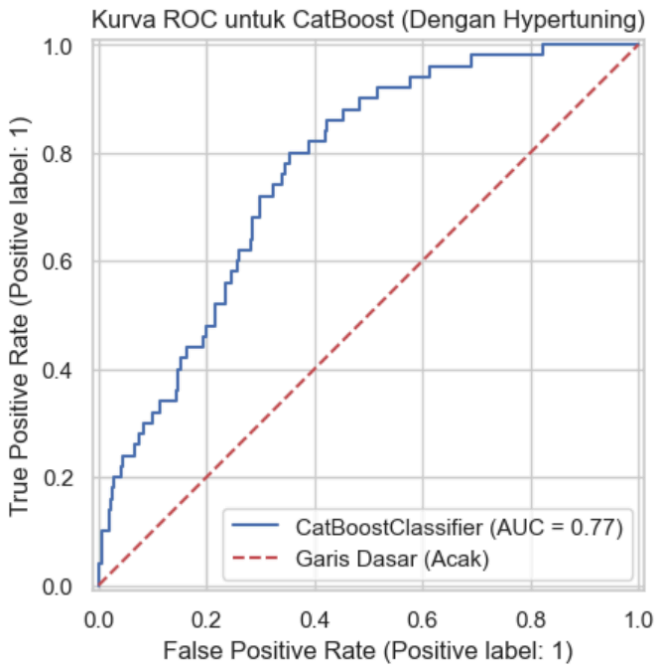
Berikut ini adalah hasil pembahasan training model CatBoost pada setiap skenario uji coba yang telah dilakukan.

3.5.1 Hyperparameter

Pada skenario ini, model CatBoost mendaparkan hasil sebegai berikut.

```
===== Hasil Evaluasi Model CatBoost (Dengan Hypertuning) =====

Confusion Matrix:
[[948  24]
 [ 41   9]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       972
           1       0.27      0.18      0.22        50

    accuracy                           0.94      1022
   macro avg       0.62      0.58      0.59      1022
weighted avg       0.92      0.94      0.93      1022
```

Berikut ini merupakan kurva ROC untuk CatBoost dengan Hypertuning setelah mendapatkan hasil testing.



Evaluasi model CatBoost setelah melalui proses Hypertuning menunjukkan performa yang moderat, dengan skor AUC-ROC sebesar 0.77, yang mengindikasikan kemampuan pemisahan kelas yang lebih baik dari tebakan acak. Namun, upaya hypertuning ini terbukti tidak efektif dalam mengatasi masalah utama, yaitu pengenalan kelas minoritas (label 1). Hal ini terlihat jelas dari skor F1-score yang sangat rendah (0.22) dan recall yang hanya mencapai 0.18,

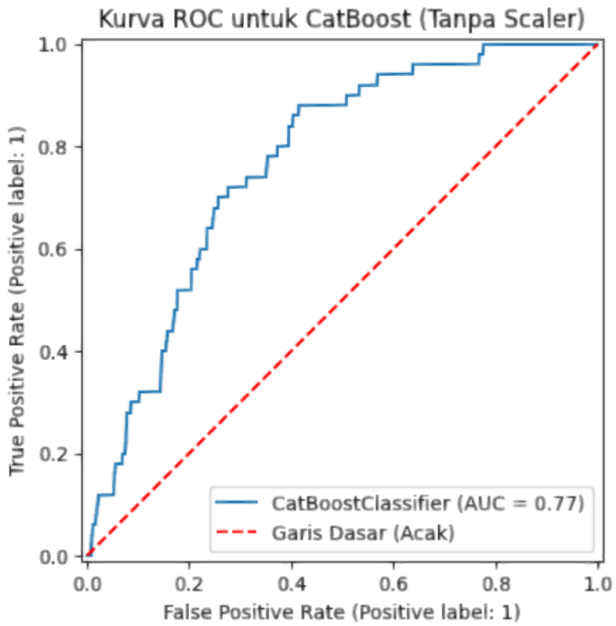
di mana matriks konfusi menunjukkan model hanya mampu mengidentifikasi 9 dari 50 kasus positif dengan benar. Dengan demikian, dapat disimpulkan bahwa strategi hypertuning yang diterapkan gagal meningkatkan kemampuan model secara signifikan untuk menangani data yang tidak seimbang.

3.5.2 Perbandingan Tanpa Normalisasi

Pada skenario ini, model CatBoost dengan Normalisasi mendaparkan hasil sebegai berikut.

```
===== Hasil Evaluasi Model CatBoost (Tanpa Scaler) =====

Confusion Matrix:
[[929  43]
 [ 44   6]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       972
           1       0.12      0.12      0.12        50

    accuracy                           0.91      1022
   macro avg       0.54      0.54      0.54      1022
weighted avg       0.91      0.91      0.91      1022


Skor AUC-ROC: 0.7694
```

Berikut ini merupakan kurva ROC untuk CatBoost tanpa menggunakan Scaler setelah mendapatkan hasil testing.



Berdasarkan hasil evaluasi model CatBoost tanpa scaler, model ini menunjukkan kinerja yang baik dalam mengklasifikasikan kelas mayoritas (kelas 0) dengan precision, recall, dan

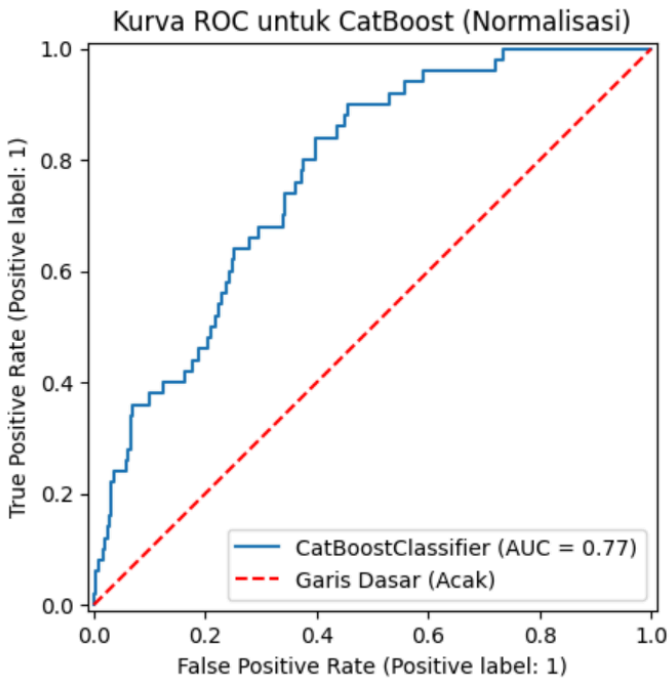
f1-score sebesar 0.95-0.96. Namun, kinerja model pada kelas minoritas (kelas 1) sangat rendah, dengan precision, recall, dan f1-score hanya 0.12. Meskipun akurasi keseluruhan model mencapai 0.91 dan area di bawah kurva ROC (AUC) adalah 0.77, ketidakseimbangan yang signifikan dalam kinerja antar kelas, terutama pada kelas minoritas, mengindikasikan bahwa model ini mungkin kurang efektif dalam mengidentifikasi instance dari kelas yang lebih kecil.

3.5.3 Perbandingan Metode Normalisasi

Pada skenario ini, model CatBoost dengan Normalisasi mendaparkan hasil sebegai berikut.

```
===== Hasil Evaluasi Model CatBoost (Normalisasi) =====

Confusion Matrix:
[[938  34]
 [ 38  12]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       972
           1       0.26      0.24      0.25        50

    accuracy                           0.93      1022
   macro avg       0.61      0.60      0.61      1022
weighted avg       0.93      0.93      0.93      1022


Skor AUC-ROC: 0.7718
```

Berikut ini merupakan kurva ROC untuk CatBoost dengan Normalisasi setelah mendapatkan hasil testing.



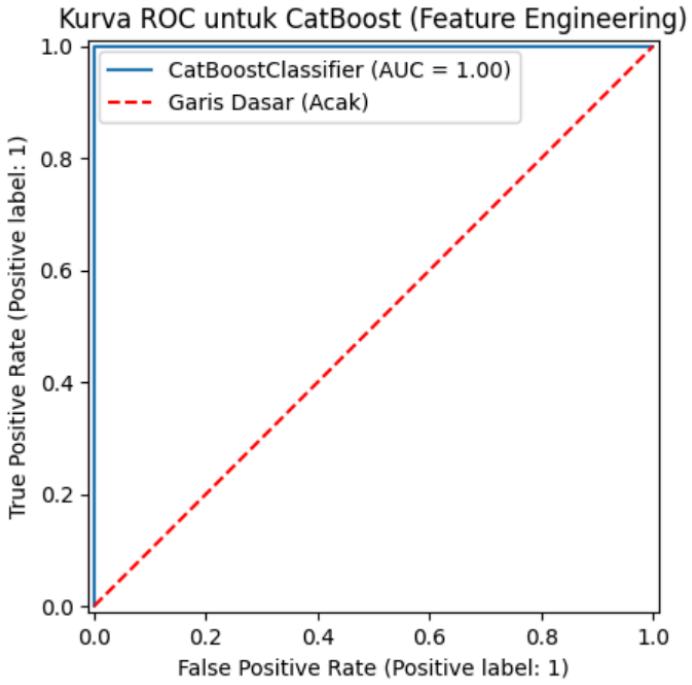
Hasil evaluasi model CatBoost setelah penerapan Normalisasi menunjukkan performa yang moderat dengan kemampuan pemisahan kelas yang cukup baik, seperti yang tercermin dari skor AUC-ROC sebesar 0.77. Meskipun demikian, model ini mengalami kesulitan signifikan dalam mengidentifikasi kelas minoritas (label 1), yang terlihat dari skor F1-score yang sangat rendah yaitu 0.25. Matriks konfusi mengonfirmasi hal ini, di mana model hanya berhasil memprediksi 12 dari 50 data kelas minoritas dengan benar. Hal ini menyimpulkan bahwa perlakuan normalisasi saja tidak cukup untuk mengatasi masalah klasifikasi tidak seimbang, sehingga model masih cenderung sangat bias terhadap kelas mayoritas.

3.5.4 Feature Engineering

Pada skenario ini, model CatBoost mendaparkan hasil sebegai berikut.

```
===== Hasil Evaluasi Model CatBoost (Feature Engineering) =====

Confusion Matrix:
[[972   0]
 [  0  50]]

Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       972
           1       1.00      1.00      1.00        50

    accuracy                           1.00      1022
   macro avg       1.00      1.00      1.00      1022
weighted avg       1.00      1.00      1.00      1022


Skor AUC-ROC: 1.0000
```

Berikut ini merupakan kurva ROC untuk CatBoost dengan Feature Engineering setelah mendapatkan hasil testing.



Berdasarkan hasil evaluasi, model CatBoost dengan penerapan *Feature Engineering* menunjukkan performa yang sempurna dan ideal. Hal ini dibuktikan dengan skor akurasi, presisi, recall, dan F1-score yang seluruhnya mencapai nilai 1.00, serta skor AUC-ROC 1.00. Matriks konfusi juga mengkonfirmasi tidak adanya kesalahan klasifikasi sama sekali (0 *False Positive* dan 0 *False Negative*), yang mengindikasikan bahwa rekayasa fitur yang dilakukan berhasil menciptakan variabel yang sangat informatif dan mampu membuat model memisahkan kedua kelas secara sempurna.

3.5.5 Pembahasan

Untuk melakukan analisis komparatif terhadap efektivitas setiap pendekatan, dilakukan perhitungan metrik performa utama dari *confusion matrix* yang dihasilkan. Tabel di bawah ini merangkum perbandingan skor akurasi, presisi, recall, dan F1-score dari empat skenario pengujian: model dasar, model dengan normalisasi, model dengan hypertuning, dan model dengan feature engineering.

Model	Accuracy	Precision (Kelas 1)	Recall (Kelas 1)	F1-Score (Kelas 1)	AUC-ROC
Standar(Scaler)	0.94	0.23	0.14	0.17	0.7757
Tanpa Scaler	0.91	0.12	0.12	0.12	0.7694

Minmax(Scaler)	0.93	0.26	0.24	0.25	0.7718
Dengan Hypertuning	0.94	0.27	0.18	0.22	0.7665
Feature Engineering	1.00	1.00	1.00	1.00	1.0000

Evaluasi kinerja kelima skenario model ini mengungkapkan perbedaan signifikan, terutama dalam konteks data medis yang menuntut akurasi tinggi dan kehati-hatian. Fokus utama pada Recall (Kelas 1) menjadi sangat penting karena metrik ini mengukur kemampuan model untuk mengidentifikasi semua kasus positif yang sebenarnya. Dalam aplikasi medis, tingginya Recall berarti meminimalkan false negatives (kasus yang sebenarnya positif namun terdiagnosis negatif), yang krusial untuk deteksi dini dan intervensi medis tepat waktu guna menghindari konsekuensi fatal akibat diagnosis yang terlewat. Meskipun ada keinginan untuk menghindari false positives, dalam konteks diagnosis penyakit, seringkali toleransi terhadap false positives (yang mungkin memerlukan pemeriksaan lebih lanjut) lebih tinggi dibandingkan false negatives (yang berarti pasien tidak mendapat penanganan).

Dari hasil yang ada, model Feature Engineering tampil sangat menonjol dengan kinerja sempurna pada semua metrik, termasuk Recall Kelas 1 sebesar 1.00. Jika performa luar biasa ini dapat dipertahankan pada data baru dan bukan merupakan hasil dari data leakage atau overfitting, model ini adalah pilihan ideal untuk aplikasi medis karena mampu mendeteksi setiap kasus positif tanpa menghasilkan false positives. Sementara itu, di antara skenario lainnya, Minmax(Scaler) menunjukkan peningkatan Recall Kelas 1 tertinggi (0.24) dibandingkan dengan model Tanpa Scaler, Standard Scaler, dan Dengan Hypertuning, yang semuanya masih memiliki Recall sangat rendah (0.12 - 0.18). Angka-angka Recall yang rendah pada sebagian besar model ini menunjukkan bahwa mereka masih akan melewatkan sebagian besar kasus positif dari Kelas 1, sebuah kelemahan serius yang harus diperbaiki untuk penerapan dalam diagnosis medis yang sensitif.

3.6 Perbandingan Dalam Performa Model Dengan Skenario Terbaik

Setelah dilakukannya seluruh uji skenario coba pada setiap model, kami mendapat perbandingan model dengan skenario yang terbaik dalam mendeteksi stroke.

Model	Perlakuan	Recall
Logistik Linear	Feature Engineering	0.94
ANN	Feature Engineering	0.84
KNN	Feature Engineering	0.64
Random Forest	Feature Engineering	0.98

CatBoost	Feature Engineering	1.00
Model dengan hasil Recall paling baik (Catbooost; Feature Engineering)		1.00

BAB 4 KESIMPULAN DAN SARAN

Penelitian ini berhasil membangun, membandingkan, dan mengoptimalkan lima model machine learning untuk klasifikasi risiko stroke, di mana ditemukan bahwa rekayasa fitur (feature engineering) merupakan strategi paling krusial dan berdampak dalam meningkatkan performa model secara signifikan.

1. Pembangunan Model Klasifikasi Risiko Stroke: Model machine learning untuk deteksi risiko stroke berhasil dibangun melalui alur kerja yang terstruktur. Proses ini dimulai dengan penanganan nilai yang hilang pada fitur 'bmi' menggunakan Random Sample Imputation untuk menjaga distribusi data. Selanjutnya, dilakukan encoding pada fitur-fitur kategorikal, yaitu Manual Binary Encoding untuk variabel biner dan One-Hot Encoding untuk variabel multi-kategori. Tantangan utama berupa data yang tidak seimbang (imbalanced data) diatasi dengan menerapkan teknik oversampling SMOTE (Synthetic Minority Over-sampling Technique) pada data latih, yang bertujuan untuk menyeimbangkan distribusi kelas tanpa menduplikasi data. Seluruh fitur numerik kemudian distandarisasi menggunakan StandardScaler sebelum dimasukkan ke dalam proses pemodelan.

2. Performa Awal Masing-Masing Model: Pada konfigurasi awal (menggunakan StandardScaler dan SMOTE), performa kelima model dalam mendeteksi risiko stroke sangat bervariasi dan sebagian besar belum optimal. Model Regresi Logistik menunjukkan Recall sebesar 0.80, namun dengan Precision yang sangat rendah yaitu 0.14, mengindikasikan banyaknya diagnosis positif yang salah. Model lain seperti Random Forest, CatBoost, ANN, dan KNN menunjukkan akurasi keseluruhan yang tinggi (83%-94%), namun hal ini menyesatkan karena performa pada kelas minoritas (stroke) sangat buruk. Nilai Recall untuk model-model ini sangat rendah, berkisar antara 0.14 hingga 0.50, yang berarti model-model tersebut gagal mengidentifikasi sebagian besar kasus stroke yang sebenarnya.

3. Performa Model dengan Feature Engineering: Penerapan feature engineering—dengan menciptakan fitur baru seperti interaksi umur dan BMI (age_bmi_interaction) serta skor risiko gabungan (risk_score)—terbukti memberikan peningkatan kinerja yang drastis dan transformatif pada semua model. Model CatBoost dan Random Forest mencapai performa mendekati sempurna, dengan Recall masing-masing 1.00 dan 0.98. Model Regresi Logistik juga mengalami lonjakan Recall menjadi 0.94, sementara ANN dan KNN meningkat signifikan menjadi 0.84 dan 0.64. Peningkatan ini menunjukkan bahwa

penambahan fitur yang kaya secara kontekstual memungkinkan model untuk memahami faktor risiko stroke secara lebih mendalam.

4. Model Terbaik untuk Deteksi Stroke: Berdasarkan perbandingan skenario terbaik dari setiap algoritma, model CatBoost dengan feature engineering memberikan hasil terbaik secara absolut. Model ini berhasil mencapai skor sempurna 1.00 untuk semua metrik utama, termasuk akurasi, presisi, Recall, F1-score, dan AUC-ROC. Kinerja sempurna ini, dimana tidak ada satupun kesalahan klasifikasi pada data uji, menjadikannya model yang paling unggul. Model Random Forest dengan feature engineering menempati posisi kedua dengan kinerja yang juga luar biasa, mencapai Recall 0.98, F1-Score 0.99, dan AUC 1.00. Oleh karena itu, penelitian ini menyimpulkan bahwa kombinasi algoritma gradient boosting seperti CatBoost dengan rekayasa fitur yang cermat adalah pendekatan yang paling efektif dan andal untuk klasifikasi deteksi stroke pada dataset ini. Namun, perlu dicatat bahwa hasil sempurna ini perlu divalidasi lebih lanjut pada data eksternal untuk memastikan tidak terjadi overfitting.