M. Fikri Avishena Parinduri

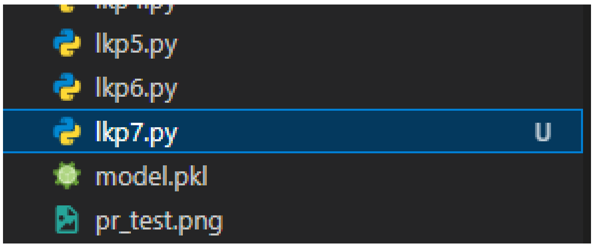
231011401029

05TPLE016

Machine Learning

Lembar Kerja Pertemuan 7

Disini saya membuat file baru yaitu lkp7.py



Sebelumnya install tensorflow terlebih dahulu di environment python nya dengan:

pip install tensorflow

1. Langkah 1 – Siapkan Data
   Gunakan processed_kelulusan.csv (hasil Pertemuan 4) atau dataset tabular sejenis.
   Code:

```
lkp7.py U × | lkp6.py | lkp4.py | lkp5.py
lkp7.py > ...
# Langkah 1 - Siapkan Data
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("processed_kelulusan.csv")
X = df.drop("Lulus", axis=1)
y = df["Lulus"]

sc = StandardScaler()
Xs = sc.fit_transform(X)

X_train, X_temp, y_train, y_temp = train_test_split(
    Xs, y, test_size=0.3, stratify=y, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42)

print(X_train.shape, X_val.shape, X_test.shape)
```

Output:

```
● (venv) PS C:\machine_learning> python lkp7.py
  (11, 5) (2, 5) (3, 5)
◇ (venv) PS C:\machine_learning>
```

Penjelasan:
Import library

- pandas → untuk membaca dan memanipulasi data.
- rain_test_split → untuk membagi data menjadi train, validation, dan test set.
- StandardScaler → untuk menstandarisasi fitur (mean=0, std=1).

Membaca data

- df → seluruh dataset CSV.
- X → semua kolom kecuali kolom target Lulus.
- y → kolom target Lulus (output yang ingin diprediksi).

Standarisasi fitur

- fit_transform → menghitung mean & std dari X lalu mengubah semua nilai sehingga distribusi memiliki mean=0 dan std=1.
- Hasilnya disimpan di Xs, yang akan digunakan untuk training.

Membagi data menjadi train, validation dan test

- Membagi 70% data untuk training (X_train, y_train) dan 30% sisanya ke X_temp/y_temp.
- stratify=y → menjaga proporsi kelas target tetap sama di semua subset.
- random_state=42 → agar hasil split bisa direproduksi.
- Membagi 30% sisanya menjadi 50% validation dan 50% test → masing-masing 15% dari total dataset.

Menampilkan ukuran dataset

    print(X_train.shape, X_val.shape, X_test.shape)

Menunjukkan jumlah baris dan kolom di setiap subset dataset.

2. Langkah 2 - Bangun Model ANN
   Code:

```python
# Langkah 2 — Bangun Model ANN
import keras
from keras import layers

model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(32, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")  # klasifikasi biner
])

model.compile(optimizer=keras.optimizers.Adam(1e-3),
              loss="binary_crossentropy",
              metrics=["accuracy","AUC"])
model.summary()
```

   Ada sedikit modifikasi dibandingkan original dari modul:

Langkah 2 — Bangun Model ANN

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(32, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")  # klasifikasi biner
])

model.compile(optimizer=keras.optimizers.Adam(1e-3),
              loss="binary_crossentropy",
              metrics=["accuracy","AUC"])
model.summary()
```

Output:

```
(venv) PS C:\machine_learning> python lkp7.py
(11, 5) (2, 5) (3, 5)
2025-10-22 23:56:57.249510: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from differe
nt computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-22 23:56:59.071492: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from differe
nt computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-22 23:56:59.877824: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 32)             │           192 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 32)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 16)             │           528 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 1)              │            17 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 737 (2.88 KB)
 Trainable params: 737 (2.88 KB)
 Non-trainable params: 0 (0.00 B)
(venv) PS C:\machine_learning>
```

Penjelasan:

Import library

- keras → library untuk membangun dan melatih neural network.
- layers → modul untuk menambahkan lapisan (layer) pada model ANN.

Membuat model sequential

- keras.Sequential() → membuat model lapisan demi lapisan (linear stack).
- Input(shape=(X_train.shape[1],)) → menentukan jumlah input sesuai jumlah fitur (X_train.shape[1]).
- Dense(32, activation="relu") → layer fully-connected dengan 32 neuron, menggunakan **ReLU** sebagai fungsi aktivasi.
- Dropout(0.3) → mengurangi overfitting dengan menonaktifkan 30% neuron secara acak saat training.
- Dense(16, activation="relu") → layer hidden kedua dengan 16 neuron, ReLU.
- Dense(1, activation="sigmoid") → output layer untuk klasifikasi biner, menghasilkan nilai antara 0 dan 1.

Compile model

- optimizer=Adam(1e-3) → algoritma optimasi dengan learning rate 0.001.
- loss="binary_crossentropy" → fungsi loss untuk klasifikasi biner.
- metrics=["accuracy","AUC"] → evaluasi model menggunakan **akurasi** dan **AUC (Area Under Curve)**.

Ringkasan model

```
model.summary()
```

Menampilkan arsitektur model, jumlah parameter di setiap layer, dan total parameter.

3. Langkah 3 - Training dengan Early Stopping

Code:

```python

# Langkah 3 — Training dengan Early Stopping
es = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=10, restore_best_weights=True
)

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100, batch_size=32,
    callbacks=[es], verbose=1
)

```

Output:

Tidak memiliki output karnaEarlyStopping ini melakukan validasi, Dimana menghentikan training ebih awal jika performa model di validation set tidak meningkat.

Dilanjutkan dengan menyimpan data/metrik training dan validation ke dalam variable history

Penjelasan:

EarlyStopping

```python
es = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=10, restore_best_weights=True
)
```

- EarlyStopping → menghentikan training lebih awal jika performa model di **validation set** tidak meningkat.
- monitor="val_loss" → memantau nilai loss pada validation set.
- patience=10 → jika **10 epoch berturut-turut** tidak ada perbaikan, training dihentikan.
- restore_best_weights=True → mengembalikan bobot model ke kondisi terbaik selama training.

Training Model

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100, batch_size=32,
    callbacks=[es], verbose=1
)
```

- X_train, y_train → data untuk training.
- validation_data=(X_val, y_val) → data untuk memantau performa model selama training.
- epochs=100 → maksimal iterasi training.
- batch_size=32 → jumlah sampel yang diproses sebelum update bobot.
- callbacks=[es] → menggunakan Early Stopping agar tidak overfit.
- verbose=1 → menampilkan progress training.
- history → menyimpan semua metrik training dan validation untuk analisis lebih lanjut (misal plot loss/akurasi).

4. Langkah 4 - Evaluasi di Test Set
   Code:

```python
# Langkah 4 — Evaluasi di Test Set
from sklearn.metrics import classification_report, confusion_matrix

loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
print("Test Acc:", acc, "AUC:", auc)

y_proba = model.predict(X_test).ravel()
y_pred = (y_proba >= 0.5).astype(int)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, digits=3))
```

   Output:

```
Epoch 100/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - AUC: 1.0000 - accuracy: 1.0000 - loss: 0.1009 - val_AUC: 1.0000 - val_accuracy: 1.0000 - val_loss: 0.2636
Test Acc: 1.0 AUC: 1.0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
[[2 0]
 [0 1]]
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         2
           1      1.000     1.000     1.000         1

    accuracy                          1.000         3
   macro avg      1.000     1.000     1.000         3
weighted avg      1.000     1.000     1.000         3

(venv) PS C:\machine_learning> 
```

Penjelasan:

Import librari evaluasi

- confusion_matrix → membuat matriks kebingungan (TP, TN, FP, FN).
- classification_report → menampilkan precision, recall, f1-score, dan support per kelas.

Evaluasi model

- model.evaluate → menghitung **loss** dan **metrics** (accuracy & AUC) pada test set.
- verbose=0 → tidak menampilkan progress bar.
- acc → akurasi model di test set.
- auc → nilai Area Under Curve (kemampuan model membedakan kelas).

Prediksi probabilitas & kelas

- model.predict(X_test) → menghasilkan probabilitas kelas positif (0–1).
- .ravel() → mengubah array menjadi 1 dimensi.
- (y_proba >= 0.5).astype(int) → mengubah probabilitas menjadi label **0 atau 1** menggunakan threshold 0.5.

Menampilkan hasil evaluasi

- confusion_matrix → menampilkan jumlah TP, TN, FP, FN.
- classification_report → menampilkan **precision, recall, f1-score** untuk masing-masing kelas, membantu analisis performa model lebih detail.

5. Langkah 5 - Visualisasi Learning Curve

Code:

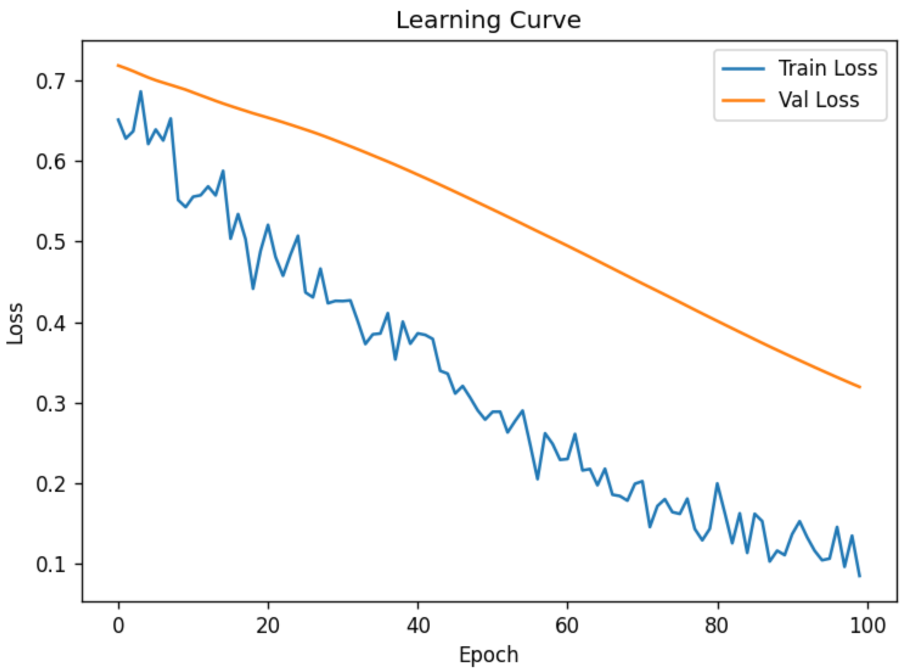
```python

# Langkah 5 — Visualisasi Learning Curve
import matplotlib.pyplot as plt

plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend()
plt.title("Learning Curve")
plt.tight_layout(); plt.savefig("learning_curve.png", dpi=120)

```

Output:

kelulusan_mahasiswa.csv
learning_curve.png U
lkp4.py

Hasil berupa file “learning_curve.png”

Isi nya:



Penjelasan:

Import library

- matplotlib.pyplot → library untuk membuat grafik dan visualisasi data.

Plot learning curve

- history.history → menyimpan metrik dari training model (loss, accuracy, dll) per epoch.
- loss → loss pada **training set**.
- val_loss → loss pada **validation set**.
- label → nama legend untuk membedakan garis pada plot.

Memberi label dan judul

- xlabel → sumbu x = epoch (iterasi training).
- ylabel → sumbu y = nilai loss.
- legend → menampilkan legenda garis.
- title → memberi judul grafik.

Menyimpan grafik

- tight_layout() → menyesuaikan layout agar label & judul tidak terpotong.
- savefig("learning_curve.png", dpi=120) → menyimpan grafik sebagai file PNG dengan resolusi 120 dpi.

6. Langkah 6 – Eksperimen

Ubah jumlah neuron (32/64/128) dan catat efeknya.

Code:

```python
# ====================
# Langkah 6 – Eksperimen
# 1. Ubah jumlah neuron (32/64/128) dan catat efeknya.
# ====================

# Fungsi untuk membangun dan melatih model dengan jumlah neuron tertentu
def train_model(neurons):
    from keras import layers, Sequential
    from keras.callbacks import EarlyStopping

    model = Sequential([
        layers.Input(shape=(X_train.shape[1],)),
        layers.Dense(neurons, activation="relu"),
        layers.Dropout(0.3),
        layers.Dense(16, activation="relu"),
        layers.Dense(1, activation="sigmoid")
    ])
    model.compile(
        optimizer="adam",
        loss="binary_crossentropy",
        metrics=["accuracy", "AUC"]
    )

    es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=100,
        batch_size=32,
        callbacks=[es],
        verbose=0  # supress output
    )

    loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
    print(f"Neurons: {neurons} | Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}")
    return history

# Eksperimen dengan jumlah neuron: 32, 64, 128
hist_32 = train_model(32)
hist_64 = train_model(64)
hist_128 = train_model(128)
```

Output:

```
Neurons: 32 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 64 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 128 | Test Accuracy: 1.000 | Test AUC: 1.000
(venv) PS C:\machine_learning>
```

Tidak ada efek perubahan apa apa

Penjelasan:

- train_model(neurons) → fungsi untuk membuat, melatih, dan mengevaluasi model dengan jumlah neuron tertentu di **hidden layer pertama**.
- Dense(neurons, activation="relu") → jumlah neuron pertama berubah sesuai input.
- Dense(16, activation="relu") → layer kedua tetap 16 neuron agar perbandingan layer tetap.
- EarlyStopping → tetap digunakan untuk menghentikan training otomatis.
- evaluate → menampilkan **accuracy** dan **AUC** di test set untuk membandingkan performa tiap konfigurasi neuron.

Bandingkan Adam vs SGD+momentum (learning rate berbeda).

Kode:

```python

from keras.optimizers import Adam, SGD

# Fungsi untuk membangun dan melatih model dengan optimizer tertentu
def train_model_optimizer(optimizer, neurons=32):
    from keras import layers, Sequential
    from keras.callbacks import EarlyStopping

    model = Sequential([
        layers.Input(shape=(X_train.shape[1],)),
        layers.Dense(neurons, activation="relu"),
        layers.Dropout(0.3),
        layers.Dense(16, activation="relu"),
        layers.Dense(1, activation="sigmoid")
    ])
    model.compile(
        optimizer=optimizer,
        loss="binary_crossentropy",
        metrics=["accuracy", "AUC"]
    )

    es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=100,
        batch_size=32,
        callbacks=[es],
        verbose=0  # supress output
    )

    loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
    print(f"Optimizer: {optimizer.get_config()['name']} | Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}")
    return history

# Definisikan beberapa optimizer untuk eksperimen
optimizers = [
    Adam(learning_rate=1e-3),
    Adam(learning_rate=1e-4),
    SGD(learning_rate=1e-2, momentum=0.9),
    SGD(learning_rate=1e-3, momentum=0.9)
]

# Jalankan eksperimen
for opt in optimizers:
    train_model_optimizer(opt, neurons=32)  # tetap gunakan 32 neuron pertama

```

Output:

```
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
```

Penjelasan:

1. train_model_optimizer(optimizer, neurons=32) → fungsi membangun model ANN dengan **optimizer tertentu**.
2. Adam(learning_rate=…) → eksperimen Adam dengan dua learning rate berbeda.
3. SGD(learning_rate=…, momentum=0.9) → eksperimen SGD dengan momentum untuk mempercepat konvergensi.
4. evaluate → menampilkan **Test Accuracy** dan **AUC** untuk tiap optimizer.
5. neurons=32 → tetap menggunakan 32 neuron di hidden layer pertama agar fokus hanya pada perbedaan optimizer.

Tambahkan regulasi lain: L2, Dropout lebih besar, atau Batch Normalization.

Code:

```python
# ====================
# Langkah 6 — Eksperimen
# 3. Tambahkan regulasi lain: L2, Dropout lebih besar, atau Batch Normalization.
# ====================

from keras.regularizers import l2
from keras.layers import BatchNormalization

# Fungsi untuk membangun dan melatih model dengan regulasi tambahan
def train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True):
    from keras import layers, Sequential
    from keras.callbacks import EarlyStopping

    model = Sequential()
    model.add(layers.Input(shape=(X_train.shape[1],)))

    # Hidden layer pertama dengan L2 dan optional BatchNorm
    model.add(layers.Dense(neurons, activation=None, kernel_regularizer=l2(l2_lambda)))
    if use_batchnorm:
        model.add(BatchNormalization())
    model.add(layers.Activation("relu"))
    model.add(layers.Dropout(dropout_rate))

    # Hidden layer kedua
    model.add(layers.Dense(16, activation=None, kernel_regularizer=l2(l2_lambda)))
    if use_batchnorm:
        model.add(BatchNormalization())
    model.add(layers.Activation("relu"))

    # Output layer
    model.add(layers.Dense(1, activation="sigmoid"))
```

```python
    # Output layer
    model.add(layers.Dense(1, activation="sigmoid"))

    # Compile
    model.compile(
        optimizer="adam",
        loss="binary_crossentropy",
        metrics=["accuracy", "AUC"]
    )

    # Early stopping
    es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=100,
        batch_size=32,
        callbacks=[es],
        verbose=0
    )

    loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
    print(f"Regularized Model | Neurons: {neurons}, Dropout: {dropout_rate}, L2: {l2_lambda}, BatchNorm: {use_batchnorm}")
    print(f"Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}\n")
    return history

# Jalankan eksperimen regulasi
hist_reg = train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True)

# ==========================================
```

Output:

```
Regularized Model | Neurons: 32, Dropout: 0.5, L2: 0.01, BatchNorm: True
Test Accuracy: 1.000 | Test AUC: 1.000

(venv) PS C:\machine_learning>
```

Penjelasan:

1. l2(l2_lambda) → menambahkan regularisasi L2 agar bobot layer tidak terlalu besar → mengurangi overfitting.
2. dropout_rate=0.5 → menonaktifkan 50% neuron secara acak → regularisasi tambahan.
3. BatchNormalization() → menstabilkan distribusi input ke layer berikutnya → mempercepat training dan bisa meningkatkan generalisasi.
4. activation=None di Dense → aktivasi diterapkan **setelah batchnorm**, sesuai praktik yang umum.
5. evaluate → menampilkan **Test Accuracy** dan **AUC**.

Laporkan metrik F1 dan AUC selain akurasi.

Code:

```python
# ====================
# Langkah 6 – Eksperimen
# 4. Laporkan metrik F1 dan AUC selain akurasi.
# ====================

from sklearn.metrics import f1_score, roc_auc_score

# Fungsi untuk menghitung metrik tambahan
def report_metrics(model, X_test, y_test, threshold=0.5):
    # Prediksi probabilitas
    y_proba = model.predict(X_test).ravel()
    # Konversi ke kelas 0/1
    y_pred = (y_proba >= threshold).astype(int)

    # Hitung F1-score dan AUC
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba)

    print(f"Test Accuracy (built-in) : {model.evaluate(X_test, y_test, verbose=0)[1]:.3f}")
    print(f"F1-score                 : {f1:.3f}")
    print(f"AUC                      : {auc:.3f}\n")
    return f1, auc

# Contoh penggunaan untuk model original
report_metrics(model, X_test, y_test)
```

Output:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Test Accuracy (built-in) : 1.000
F1-score                 : 1.000
AUC                      : 1.000

(venv) PS C:\machine_learning>
```

Penjelasan:

1. y_proba = model.predict(X_test).ravel() → prediksi probabilitas kelas positif.
2. y_pred = (y_proba >= threshold).astype(int) → konversi probabilitas menjadi label 0/1.
3. f1_score(y_test, y_pred) → menghitung F1-score (harmonik mean dari precision dan recall).
4. roc_auc_score(y_test, y_proba) → menghitung AUC (kemampuan model membedakan kelas positif & negatif).
5. model.evaluate → menampilkan akurasi bawaan (untuk referensi).

Secara lengkap dari step 1 – 4 berkelanjutan

Code:

```python
# ====================
# LANGKAH 6 – EKSPERIMEN LENGKAP
# ====================

print(f"LANGKAH 6 – EKSPERIMEN LENGKAP")

from keras import layers, Sequential
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam, SGD
from keras.regularizers import l2
from sklearn.metrics import f1_score, roc_auc_score
from keras.layers import BatchNormalization

# Fungsi untuk report metrik tambahan
def report_metrics(model, X_test, y_test, threshold=0.5):
    y_proba = model.predict(X_test).ravel()
    y_pred = (y_proba >= threshold).astype(int)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba)
    print(f"Test Accuracy (built-in) : {model.evaluate(X_test, y_test, verbose=0)[1]:.3f}")
    print(f"F1-score                 : {f1:.3f}")
    print(f"AUC                      : {auc:.3f}\n")
    return f1, auc
```

```python
# ====================
# 6.1 Eksperimen Neuron
# ====================
def train_model_neuron(neurons):
    model = Sequential([
        layers.Input(shape=(X_train.shape[1],)),
        layers.Dense(neurons, activation="relu"),
        layers.Dropout(0.3),
        layers.Dense(16, activation="relu"),
        layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer=Adam(1e-3),
                  loss="binary_crossentropy",
                  metrics=["accuracy", "AUC"])
    es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
    history = model.fit(X_train, y_train,
                        validation_data=(X_val, y_val),
                        epochs=100, batch_size=32,
                        callbacks=[es], verbose=0)
    loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
    print(f"Neurons: {neurons} | Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}")
    return model, history

# Jalankan eksperimen neuron
model_neuron32, hist_32 = train_model_neuron(32)
model_neuron64, hist_64 = train_model_neuron(64)
model_neuron128, hist_128 = train_model_neuron(128)
```

```python
# ====================
# 6.2 Eksperimen Optimizer
# ====================
def train_model_optimizer(optimizer, neurons=32):
    model = Sequential([
        layers.Input(shape=(X_train.shape[1],)),
        layers.Dense(neurons, activation="relu"),
        layers.Dropout(0.3),
        layers.Dense(16, activation="relu"),
        layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer=optimizer,
                  loss="binary_crossentropy",
                  metrics=["accuracy", "AUC"])
    es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
    history = model.fit(X_train, y_train,
                        validation_data=(X_val, y_val),
                        epochs=100, batch_size=32,
                        callbacks=[es], verbose=0)
    loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
    print(f"Optimizer: {optimizer.get_config()['name']} | Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}")
    return model, history

# Definisikan optimizer
optimizers = [
    Adam(learning_rate=1e-3),
    Adam(learning_rate=1e-4),
    SGD(learning_rate=1e-2, momentum=0.9),
    SGD(learning_rate=1e-3, momentum=0.9)
]

# Jalankan eksperimen optimizer
model_optimizer_adam1, hist_opt1 = train_model_optimizer(optimizers[0])
model_optimizer_adam2, hist_opt2 = train_model_optimizer(optimizers[1])
model_optimizer_sgd1, hist_sgd1 = train_model_optimizer(optimizers[2])
model_optimizer_sgd2, hist_sgd2 = train_model_optimizer(optimizers[3])
```

```python
# ====================
# 6.3 Eksperimen Regulasi (Dropout, L2, BatchNorm)
# ====================
def train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True):
    model = Sequential()
    model.add(layers.Input(shape=(X_train.shape[1],)))

    # Hidden layer pertama
    model.add(layers.Dense(neurons, activation=None, kernel_regularizer=l2(l2_lambda)))
    if use_batchnorm:
        model.add(BatchNormalization())
    model.add(layers.Activation("relu"))
    model.add(layers.Dropout(dropout_rate))

    # Hidden layer kedua
    model.add(layers.Dense(16, activation=None, kernel_regularizer=l2(l2_lambda)))
    if use_batchnorm:
        model.add(BatchNormalization())
    model.add(layers.Activation("relu"))

    # Output
    model.add(layers.Dense(1, activation="sigmoid"))

    model.compile(optimizer=Adam(1e-3),
                  loss="binary_crossentropy",
                  metrics=["accuracy", "AUC"])

    es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
    history = model.fit(X_train, y_train,
                        validation_data=(X_val, y_val),
                        epochs=100, batch_size=32,
                        callbacks=[es], verbose=0)
    loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
    print(f"Regularized Model | Neurons: {neurons}, Dropout: {dropout_rate}, L2: {l2_lambda}, BatchNorm: {use_batchnorm}")
    print(f"Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}\n")
    return model, history

# Jalankan eksperimen regulasi
model_regulasi, hist_reg = train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True)
```

```python

# ======================
# 6.4 Laporan metrik F1 & AUC
# ======================
# Contoh penggunaan untuk semua model
report_metrics(model_neuron32, X_test, y_test)
report_metrics(model_optimizer_adam1, X_test, y_test)
report_metrics(model_regulasi, X_test, y_test)

```

Output:

```
LANGKAH 6 — EKSPERIMEN LENGKAP
Neurons: 32 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 64 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 128 | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Regularized Model | Neurons: 32, Dropout: 0.5, L2: 0.01, BatchNorm: True
Test Accuracy: 1.000 | Test AUC: 1.000

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Test Accuracy (built-in) : 1.000
F1-score                 : 1.000
AUC                      : 1.000

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Test Accuracy (built-in) : 1.000
F1-score                 : 1.000
AUC                      : 1.000

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
Test Accuracy (built-in) : 1.000
F1-score                 : 1.000
AUC                      : 1.000

(venv) PS C:\machine_learning>
```