PERTEMUAN 07

MACHINE LEARNING

1. MEMBACA DAN MEMERIKSA DATA

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

Tujuan: memisahkan fitur (X) dan target (y).

- Lulus adalah kolom target (0 = tidak lulus, 1 = lulus).
- Sisanya (IPK, Absensi, dsb.) jadi fitur masukan model.

2. CEK DISTRIBUSI KELAS

```python
print("Distribusi kelas sebelum split:")
print(y.value_counts())
```

Tujuan: mengetahui apakah dataset seimbang (balanced) atau tidak seimbang (imbalanced).

Keseimbangan data penting, karena model bisa *bias* terhadap kelas mayoritas jika dataset timpang.

3. SPLIT DATA (Train, Validation, Test)

X_train_raw, X_temp_raw, y_train, y_temp = train_test_split(

X, y, test_size=0.3, stratify=y, random_state=42)

Data dibagi 70% train dan 30% sisanya (temp) untuk val+test.

- stratify=y artinya pembagian menjaga proporsi kelas tetap sama di setiap subset.

Kemudian 30% sisanya dibagi dua lagi (masing-masing 15%):

X_val_raw, X_test_raw, y_val, y_test = train_test_split(

X_temp_raw, y_temp, test_size=0.5, stratify=y_temp, random_state=42)

Jika kelas terlalu sedikit dan stratify gagal, program otomatis fallback ke random split agar tidak error.

4. STANDARDISASI FITUR

```python
sc = StandardScaler()
sc.fit(X_train_raw)
X_train = sc.transform(X_train_raw)
X_val = sc.transform(X_val_raw)
X_test = sc.transform(X_test_raw)
```

StandardScaler mengubah setiap fitur numerik agar punya:

- mean = 0
- standar deviasi = 1

Ini penting untuk neural network, karena membantu konvergensi lebih cepat dan stabil saat training.

Catatan penting: scaler hanya di-*fit* di data train, lalu di-*transform* di val/test untuk menghindari data leakage.

5. MEMBANGUN MODEL NEURAL NETWORK

```python
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")
])
```

Struktur jaringan:

- Input layer: sebanyak jumlah fitur (X_train.shape[1])
- Hidden layer 1: 32 neuron + ReLU activation
- Dropout(0.3): mencegah overfitting (acak 30% neuron di setiap iterasi)
- Hidden layer 2: 16 neuron + ReLU
- Output layer: 1 neuron + sigmoid → karena ini klasifikasi biner (output antara 0–1)

6. KOMPILASI MODEL

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

Penjelasan:

- Adam → optimizer adaptif (belajar otomatis menyesuaikan laju pembelajaran)
- binary_crossentropy → loss function standar untuk klasifikasi 0/1
- accuracy dan AUC → metrik evaluasi tambahan

7. EARLY STOPPING

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

Tujuan: berhenti training otomatis jika val_loss tidak membaik setelah 10 epoch. Menghindari *overfitting* (model terus belajar tapi malah memburuk di validation).

8. TRAINING MODEL

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

Melatih model dengan 70% data train dan memonitor performa di 15% data validation.

- Maksimal 100 epoch
- Tapi bisa berhenti lebih cepat karena EarlyStopping.

history menyimpan log metrik (loss, val_loss, accuracy, dll) yang nanti bisa diplot.

9. EVALUASI DI TEST SET

```python
loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f} | Test Accuracy: {acc:.4f} | Test AUC: {auc:.4f}")
```

Mengevaluasi performa di 15% data test (data yang benar-benar baru bagi model).

10. LEARNING CURVE

```python
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
```

Menampilkan grafik loss selama training:

- Kalau val_loss terus turun → model masih belajar.
- Kalau val_loss naik sementara train_loss turun → mulai *overfitting*.