LAPORAN PERTEMUAN 5 – MODELING MACHINE LEARNING

Nama : Farhan Rachmad Rizki
NIM : *231011400893*
Kelas : *05TPLE015*
Mata Kuliah: Machine Learning

Pendahuluan

Dalam penelitian kali ini dilakukan pemodelan machine learning menggunakan dataset hasil *data preparation* sebelumnya yaitu processed_kelulusan.csv.

Dataset ini berisi beberapa variabel seperti IPK, Jumlah_Absensi, Waktu_Belajar_Jam, serta fitur tambahan Rasio_Absensi dan IPK_x_Study, dengan target variabel Lulus (1 = Lulus, 0 = Tidak Lulus).

Tujuan utama dari penelitian ini adalah:

- Melakukan **pemilihan model terbaik** antara *Logistic Regression* dan *Random Forest*
- Melakukan **validasi model dan tuning hyperparameter**
- Mengevaluasi performa model menggunakan metrik seperti *F1-score*, *Confusion Matrix*, dan *ROC-AUC*
- Melakukan **deployment sederhana** menggunakan *Flask API* untuk prediksi otomatis

Langkah 1 – Import Library dan Dataset

```
[ ]: import pandas as pd
import matplotlib.pyplot as plt
import joblib
from flask import Flask, request, jsonify
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    f1_score, classification_report, confusion_matrix,
    roc_auc_score, roc_curve
)
import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv("processed_kelulusan.csv")

X = df.drop("Lulus", axis=1)
y = df["Lulus"]

try:
    X_train, X_temp, y_train, y_temp = train_test_split(
        X, 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
    )
except ValueError:
    print(" Peringatan: Stratify dinonaktifkan karena salah satu kelas terlalu sedikit.")
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

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

Penjelasan:

-Membaca dataset hasil pertemuan sebelumnya.
-Memisahkan fitur (X) dan target (y).
-Membagi data menjadi 70% train, 15% validation, 15% test.
-Jika *stratify* gagal karena data tidak seimbang, otomatis nonaktif.

Hasil:
Ukuran data `(7, 5) (1, 5) (2, 5)` menunjukkan pembagian data berhasil.

Langkah 2 – Preprocessing Data

```
num_cols = X_train.select_dtypes(include="number").columns

pre = ColumnTransformer([
    ("num", Pipeline([
        ("imp", SimpleImputer(strategy="median")),
        ("sc", StandardScaler())
    ]), num_cols),
], remainder="drop")
```

Penjelasan:

-Menentukan kolom numerik yang akan diproses.
-Menggunakan *SimpleImputer* untuk mengganti nilai kosong dengan median.
-Menstandardisasi data menggunakan *StandardScaler* agar semua fitur memiliki skala serupa.

Langkah 3 – Baseline Model (Logistic Regression)

```
pipe_lr = Pipeline([
    ("pre", pre),
    ("clf", LogisticRegression(max_iter=1000, class_weight="balanced", random_state=42))
])

pipe_lr.fit(X_train, y_train)
y_val_pred = pipe_lr.predict(X_val)

print("\n===== Baseline: Logistic Regression =====")
print("F1 Score (val):", f1_score(y_val, y_val_pred, average="macro"))
print(classification_report(y_val, y_val_pred, digits=3))
```

Penjelasan:

-Menggunakan *Logistic Regression* sebagai model dasar (*baseline*).
`-class_weight="balanced"` membantu mengatasi data tidak seimbang.
-F1-score digunakan untuk menilai keseimbangan antara precision dan recall.

Hasil:
F1 Score (val) = **1.0** → model baseline sudah bekerja sangat baik.

Langkah 4 – Model Lanjutan (Random Forest)

```
pipe_rf = Pipeline([
    ("pre", pre),
    ("clf", RandomForestClassifier(
        n_estimators=300,
        max_features="sqrt",
        class_weight="balanced",
        random_state=42
    ))
])

pipe_rf.fit(X_train, y_train)
y_val_rf = pipe_rf.predict(X_val)

print("\n===== Random Forest (default) =====")
print("F1 Score (val):", f1_score(y_val, y_val_rf, average="macro"))
```

Penjelasan:

- Menggunakan *Random Forest* dengan 300 pohon keputusan.
- Model ini lebih kompleks dan mampu menangani non-linearitas.

Hasil:
F1 Score (val) = **1.0**, menunjukkan hasil sempurna pada data validasi.

Langkah 5 – Hyperparameter Tuning

```
skf = StratifiedKFold(n_splits=4, shuffle=True, random_state=42)

param = {
    "clf__max_depth": [None, 12, 20, 30],
    "clf__min_samples_split": [2, 5, 10]
}

gs = GridSearchCV(
    pipe_rf, param_grid=param, cv=skf,
    scoring="f1_macro", n_jobs=-1, verbose=1
)

gs.fit(X_train, y_train)

print("\n===== Hasil GridSearch =====")
print("Best Params:", gs.best_params_)
print("Best CV F1:", gs.best_score_)
```

Penjelasan:

- Melakukan pencarian kombinasi parameter terbaik menggunakan *GridSearchCV*.
- Parameter yang diuji: `max_depth` dan `min_samples_split`.
- Menggunakan *F1-macro* sebagai metrik penilaian pada 4-fold cross-validation.

Hasil:
Best Params: `{'clf__max_depth': None, 'clf__min_samples_split': 2}`
Best CV F1: **1.0**

Langkah 6 – Evaluasi Akhir (Test Data)

```
best_rf = gs.best_estimator_
y_val_best = best_rf.predict(X_val)
print("Best RF F1 (val):", f1_score(y_val, y_val_best, average="macro"))

final_model = best_rf  # ubah ke pipe_lr jika baseline lebih baik
y_test_pred = final_model.predict(X_test)

print("\n===== EVALUASI AKHIR (TEST) =====")
print("F1 Score (test):", f1_score(y_test, y_test_pred, average="macro"))
print(classification_report(y_test, y_test_pred, digits=3))
print("Confusion Matrix (test):")
print(confusion_matrix(y_test, y_test_pred))
```

Penjelasan:

- Menggunakan model terbaik hasil *GridSearchCV*.
- Mengevaluasi pada data test untuk melihat generalisasi model.
- *Confusion matrix* memperlihatkan akurasi prediksi per kelas.

Hasil:

- F1 Score (test) = **1.0**
- Model memprediksi semua data test dengan benar.

Langkah 7 – ROC Curve & AUC

```python
if hasattr(final_model, "predict_proba"):
    y_test_proba = final_model.predict_proba(X_test)[:, 1]
    try:
        auc = roc_auc_score(y_test, y_test_proba)
        print("ROC-AUC (test):", auc)
    except:
        print("ROC-AUC tidak dapat dihitung (kemungkinan multi-class).")

    fpr, tpr, _ = roc_curve(y_test, y_test_proba)
    plt.figure()
    plt.plot(fpr, tpr, label=f"AUC = {auc:.3f}")
    plt.plot([0, 1], [0, 1], "k--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve (Test)")
    plt.legend()
    plt.tight_layout()
    plt.savefig("roc_test.png", dpi=120)
    plt.close()
```

Penjelasan:

- Menghitung nilai **AUC (Area Under Curve)** untuk mengukur performa model secara menyeluruh.
- ROC-AUC (test) = **1.0**, menandakan model memiliki performa sempurna.

Langkah 8 – Deployment Model

```python
joblib.dump(final_model, "model.pkl")
print("\n Model tersimpan ke 'model.pkl'")

app = Flask(__name__)
MODEL = joblib.load("model.pkl")

@app.route("/predict", methods=["POST"])
def predict():
    data = request.get_json(force=True)
    X_input = pd.DataFrame([data])
    y_pred = MODEL.predict(X_input)[0]

    proba = None
    if hasattr(MODEL, "predict_proba"):
        proba = float(MODEL.predict_proba(X_input)[:, 1][0])

    return jsonify({
        "prediction": int(y_pred),
        "proba": proba
    })

if __name__ == "__main__":
    app.run(port=5000)
```

Penjelasan:

- Menyimpan model terbaik dalam file `model.pkl`.
- Membuat API sederhana menggunakan **Flask** untuk menerima input JSON dan mengembalikan hasil prediksi.
- API berjalan di `http://127.0.0.1:5000` dengan endpoint `/predict`.

Hasil:
Server berjalan sukses dan siap menerima permintaan prediksi.

Tahap Hasil dan Evaluasi Model

```
 Peringatan: Stratify dinonaktifkan karena salah satu kelas terlalu sedikit.
(7, 5) (1, 5) (2, 5)

===== Baseline: Logistic Regression =====
F1 Score (val): 1.0
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         1

    accuracy                         1.000         1
   macro avg      1.000     1.000     1.000         1
weighted avg      1.000     1.000     1.000         1


===== Random Forest (default) =====
F1 Score (val): 1.0
Fitting 4 folds for each of 12 candidates, totalling 48 fits

===== Hasil GridSearch =====
Best Params: {'clf__max_depth': None, 'clf__min_samples_split': 2}
Best CV F1: 1.0
Best RF F1 (val): 1.0

===== EVALUASI AKHIR (TEST) =====
F1 Score (test): 1.0
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         1
           1      1.000     1.000     1.000         1

    accuracy                         1.000         2
   macro avg      1.000     1.000     1.000         2
weighted avg      1.000     1.000     1.000         2

Confusion Matrix (test):
[[1 0]
 [0 1]]
ROC-AUC (test): 1.0
```

```
Model tersimpan ke 'model.pkl'
* Serving Flask app '__main__'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSG
I server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Analisis Hasil

1. **Peringatan Stratify**
   Stratify dinonaktifkan karena salah satu kelas terlalu sedikit.
   Pembagian data berhasil dengan ukuran: (7, 5) (1, 5) (2, 5).
2. **Baseline Logistic Regression**
   F1 Score (val) = 1.0
   Model mampu mengenali semua data validasi dengan sempurna.
3. **Random Forest (Default)**
   F1 Score (val) = 1.0
   Model juga bekerja sempurna sebelum dilakukan tuning.
4. **Grid Search (Hyperparameter Tuning)**
   Best Params = max_depth=None, min_samples_split=2
   Best CV F1 = 1.0
   Parameter terbaik ditemukan dengan hasil sempurna di semua fold.
5. **Evaluasi Akhir (Test Data)**
   F1 Score (test) = 1.0
   Confusion Matrix = [[1 0], [0 1]]
   Semua data test diprediksi 100% benar.
6. **ROC Curve & AUC**
   ROC-AUC = 1.0 → model mampu membedakan kelas Lulus dan Tidak Lulus secara sempurna.
   Namun hasil ini bisa terlalu ideal karena jumlah data sedikit (potensi overfitting).

7. **Penyimpanan dan Deployment**
   Model terbaik disimpan sebagai **model.pkl** dan dijalankan melalui **Flask API** di http://127.0.0.1:5000/predict.

Kesimpulan Singkat

Model **Random Forest** dengan parameter terbaik (max_depth=None, min_samples_split=2) menghasilkan performa sempurna dengan **F1 Score dan AUC = 1.0**.
Model berhasil disimpan dan dapat digunakan untuk prediksi melalui **API Flask**.
Perlu uji lanjut dengan dataset lebih besar untuk memastikan model tidak *overfitting*.