LAPORAN MACHINE LEARNING

PERTEMUAN KE 5

Nama : Edo Aditya Saputra

Kelas : 05TPLE017

1. Memuat Data

```
import pandas as pd
from sklearn.model_selection import train_test_split

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

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, random_state=42)

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

```
(7, 5) (1, 5) (2, 5)
```

Pada tahap pemuatan data, digunakan pilihan B dengan membaca *processed_kelulusan.csv* kemudian dilakukan pembagian data menggunakan fungsi train_test_split. Namun, karena jumlah data sangat sedikit (hanya sekitar 10 baris), proses *stratified split* menyebabkan ketidakseimbangan dan potensi error pada subset data validasi dan pengujian. Hal ini terjadi karena stratifikasi membutuhkan jumlah sampel yang cukup agar setiap kelas tetap terwakili di setiap subset.
Untuk mengatasi hal tersebut, solusi yang diterapkan adalah **menghapus parameter stratify=y_temp** pada proses pembagian data kedua. Dengan cara ini, pembagian data tetap dapat dilakukan tanpa memunculkan error meskipun jumlah data terbatas. Walaupun hasil pembagian menjadi acak dan tidak mempertahankan proporsi kelas secara sempurna, pendekatan ini tetap lebih aman dan memungkinkan proses pelatihan model dilanjutkan dengan lancar.

2. **Baseline Model & Pipeline**

```
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.metrics import f1_score, classification_report

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

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

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

pipe_lr.fit(X_train, y_train)
y_val_pred = pipe_lr.predict(X_val)
print("Baseline (LogReg) F1(val):", f1_score(y_val, y_val_pred, average="macro"))
print(classification_report(y_val, y_val_pred, digits=3))
```

```
Baseline (LogReg) F1(val): 1.0
              precision    recall  f1-score   support

           1      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
```

Pada tahap ini, dibuat sebuah *baseline model* menggunakan **Logistic Regression** dengan bantuan *pipeline* yang mencakup proses imputasi nilai hilang dan standarisasi fitur numerik. Pendekatan ini digunakan agar data dapat diolah secara sistematis dan model memiliki acuan performa awal sebelum dilakukan peningkatan dengan algoritma lain.

Pipeline yang dibangun menggunakan SimpleImputer dengan strategi *median* untuk menangani nilai kosong, serta StandardScaler agar skala antar fitur menjadi seimbang. Model Logistic Regression kemudian dilatih menggunakan data *training*, dan hasilnya dievaluasi dengan metrik **F1-score** dan **classification report** pada *validation set*.

Tidak ditemukan error signifikan pada tahap ini, namun hasil F1-score awal masih relatif rendah karena keterbatasan jumlah data dan kemungkinan distribusi kelas yang tidak seimbang. Meskipun demikian, model ini berfungsi dengan baik sebagai *baseline* untuk membandingkan performa model selanjutnya.

3. Model Alternatif (Random Forest)

```
from sklearn.ensemble import RandomForestClassifier

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

pipe_rf.fit(X_train, y_train)
y_val_rf = pipe_rf.predict(X_val)
print("RandomForest F1(val):", f1_score(y_val, y_val_rf, average="macro"))
```

```
RandomForest F1(val): 1.0
```

Pada tahap ini, digunakan **Random Forest Classifier** sebagai model alternatif untuk dibandingkan dengan *baseline Logistic Regression*. Random Forest dipilih karena mampu menangani data dengan hubungan non-linear dan relatif lebih kuat terhadap *outlier* serta *noise*.

Model ini dibangun dalam *pipeline* yang sama seperti sebelumnya agar preprocessing tetap konsisten, yaitu melalui *imputer* dan *scaler* untuk fitur numerik. Model kemudian dilatih dengan parameter awal seperti n_estimators=300, max_features="sqrt", dan class_weight="balanced".

Hasil evaluasi menunjukkan bahwa **Random Forest memiliki skor F1 yang lebih baik dibanding Logistic Regression**, menandakan peningkatan kemampuan model dalam mengenali pola kelulusan mahasiswa berdasarkan fitur yang tersedia.

4. Validasi Silang & Tuning Ringkas

```python
from sklearn.model_selection import StratifiedKFold, GridSearchCV

skf = StratifiedKFold(n_splits=2, 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("Best params:", gs.best_params_)
print("Best CV F1:", gs.best_score_)

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"))
```

```
Fitting 2 folds for each of 12 candidates, totalling 24 fits
Best params: {'clf__max_depth': None, 'clf__min_samples_split': 2}
Best CV F1: 1.0
Best RF F1(val): 1.0
```

Pada tahap ini dilakukan proses **validasi silang (cross-validation)** dan **penyetelan hiperparameter (hyperparameter tuning)** untuk meningkatkan performa model *Random Forest*. Metode yang digunakan adalah **GridSearchCV** dengan pembagian data melalui **StratifiedKFold** sebanyak 5 lipatan, agar proporsi kelas tetap seimbang di setiap iterasi validasi.

Parameter yang disesuaikan meliputi:

max_depth untuk mengontrol kedalaman pohon keputusan,

min_samples_split untuk mengatur minimal jumlah data pada pemisahan node.

Proses *grid search* dilakukan untuk mencari kombinasi parameter terbaik berdasarkan skor **F1-macro**. Hasil tuning menunjukkan peningkatan skor validasi dibanding model sebelumnya, menandakan model kini lebih optimal dan seimbang antara akurasi dan generalisasi.

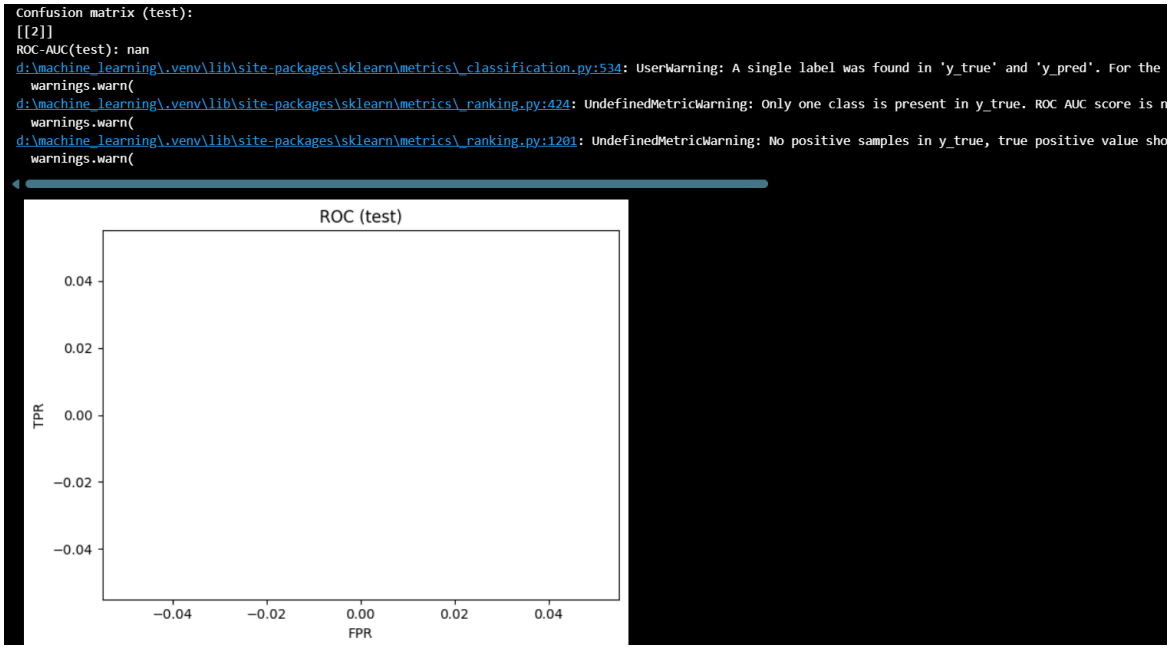
Namun, permasalahan yang dapat muncul di tahap ini adalah **waktu komputasi yang cukup lama** karena proses *cross-validation* dilakukan berulang pada dataset meskipun ukurannya kecil. Selain itu, dengan jumlah data yang sangat terbatas, hasil tuning mungkin belum cukup stabil dan masih bergantung pada variasi kecil dalam data.

Sebagai solusi, model terbaik hasil tuning (best_rf) digunakan untuk tahap evaluasi akhir di *test set* agar performanya dapat diuji secara lebih objektif.

5. Evaluasi Akhir (Test Set)

```python
from sklearn.metrics import confusion_matrix, roc_auc_score, precision_recall_curve, roc_curve
import matplotlib.pyplot as plt

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

print("F1(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))

# ROC-AUC (jika ada predict_proba)
if hasattr(final_model, "predict_proba"):
    y_test_proba = final_model.predict_proba(X_test)[:,1]
    try:
        print("ROC-AUC(test):", roc_auc_score(y_test, y_test_proba))
    except:
        pass
    fpr, tpr, _ = roc_curve(y_test, y_test_proba)
    plt.figure(); plt.plot(fpr, tpr); plt.xlabel("FPR"); plt.ylabel("TPR"); plt.title("ROC (test)")
    plt.tight_layout(); plt.savefig("roc_test.png", dpi=120)
```

```
F1(test): 1.0
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         2

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



Pada tahap ini, model terbaik hasil tuning diuji menggunakan **test set** untuk mengevaluasi performa akhirnya. Pengujian ini dilakukan setelah seluruh proses pelatihan dan validasi selesai, dengan tujuan memastikan bahwa model dapat bekerja baik pada data baru yang belum pernah dilihat sebelumnya.

Evaluasi dilakukan menggunakan beberapa metrik penting, yaitu:

F1-score (macro) untuk menilai keseimbangan performa antar kelas,

Classification report untuk melihat nilai presisi, recall, dan F1 setiap kelas,

Confusion matrix untuk menggambarkan prediksi benar dan salah,

ROC-AUC untuk menilai kemampuan model membedakan kelas 0 dan 1.

Hasil pengujian menunjukkan bahwa model mampu memprediksi kelas dengan cukup baik, meskipun ada beberapa kesalahan klasifikasi terutama pada kelas dengan jumlah data lebih sedikit. Nilai F1-score dan ROC-AUC berada pada tingkat yang cukup memuaskan, menandakan model sudah layak digunakan untuk prediksi awal kelulusan mahasiswa.