Nama : Firman Ramadhan Saputra

NIM : 231011400891

Kelas : 05TPLE015

Mata Kuliah : Machine Learning

LAPORAN LEMBAR KERJA PERTEMUAN 6 — RANDOM FOREST UNTUK KLASIFIKASI

1. **Langkah 1 – Persiapan Data**
   Gunakan Python versi 3.10.x dengan scikit-learn, pandas, matplotlib, seaborn, dan joblib. Untuk datanya, menggunakan processed_kelulusan.csv (dari Pertemuan 4) dengan kolom target Lulus.

2. **Langkah 2 – Muat data (dengan opsi A menggunakan processed_kelulusan.csv)**

   Data processed_kelulusan.csv dimuat, kolom Lulus dijadikan target, sisanya sebagai fitur. Dataset lalu dibagi menjadi train (70%), validation (15%), dan test (15%) menggunakan train_test_split dengan stratifikasi label untuk menjaga proporsi kelas.

```python
# Langkah 1 - MUAT 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"]

# split 70/15/15
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.30, stratify=y, random_state=42
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.50, stratify=y_temp, random_state=42
)
print("Hasil dari Split Muat Data: \n", X_train.shape, X_val.shape, X_test.shape)
```

   Dengan output berikut:

```
Hasil dari Split Muat Data:
 (53, 5) (12, 5) (12, 5)
```

   Interpretasi Output

- **(53, 5)** → Training set berisi **53 baris** dan **5 fitur**.
- **(12, 5)** → Validation set berisi **12 baris** dan **5 fitur**.
- **(12, 5)** → Test set berisi **12 baris** dan **5 fitur**.

Total data = 53 + 12 + 12 = **77 sampel**

- 70% = 53 sampel (train)
- 15% = 12 sampel (validation)
- 15% = 12 sampel (test)

3. Langkah 3 - Pipeline & Baseline Random Forest

Langkah ke 3 ini membuat sebuah pipeline untuk model **Random Forest** yang dilengkapi preprocessing otomatis. Pada tahap awal, kolom numerik diproses dengan **SimpleImputer** untuk mengisi nilai yang hilang menggunakan median, lalu distandarisasi dengan **StandardScaler**. Data yang sudah diproses kemudian digunakan oleh **RandomForestClassifier** dengan 300 pohon, pengaturan max_features="sqrt", serta class_weight="balanced" agar penanganan kelas tetap seimbang. Pipeline ini dilatih menggunakan data training dan divalidasi pada data validasi. Kinerja model kemudian dievaluasi menggunakan **F1-score** (macro) serta **classification report** yang menampilkan precision, recall, dan f1-score tiap kelas.

```
# Langkah 2 - Pipeline dan baseline random forest
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
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")

rf = RandomForestClassifier(
    n_estimators=300, max_features="sqrt",
    class_weight="balanced", random_state=42
)

pipe = Pipeline([("pre", pre), ("clf", rf)])
pipe.fit(X_train, y_train)

y_val_pred = pipe.predict(X_val)
print("Baseline RF - F1(Val): ", f1_score(y_val, y_val_pred, average="macro"))
print(classification_report(y_val, y_val_pred, digits=3))
```

Dengan outputnya:

```
Baseline RF - F1(Val):  1.0
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         6
           1      1.000     1.000     1.000         6


    accuracy                          1.000        12
   macro avg      1.000     1.000     1.000        12
weighted avg      1.000     1.000     1.000        12
```

Output dari code menunjukkan bahwa model **Random Forest** pada data validasi mencapai performa sempurna dengan nilai **precision, recall, dan f1-score = 1.0** untuk kedua kelas (0 dan 1). Hal ini berarti semua data positif maupun negatif berhasil diprediksi dengan benar tanpa kesalahan. Nilai **accuracy, macro average, dan weighted average** juga sama-sama 1.0, yang menandakan model memiliki kemampuan klasifikasi yang sangat baik pada dataset uji validasi ini.

4. **Langkah 4 - Validasi Silang**

Melakukan **validasi silang** menggunakan metode **Stratified K-Fold** dengan 5 lipatan untuk menjaga proporsi kelas tetap seimbang di setiap fold. Model pipeline yang sudah dibuat sebelumnya dievaluasi pada data training dengan metrik **F1-macro**. Fungsi cross_val_score menghitung skor F1 di setiap fold, kemudian hasil rata-rata (**mean**) dan penyebarannya (**standar deviasi**) ditampilkan. Dengan cara ini, performa model dapat dinilai lebih stabil dan tidak hanya bergantung pada satu pembagian data.

```python
# langkah 3 - validasi silang
from sklearn.model_selection import StratifiedKFold, cross_val_score

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(pipe, X_train, y_train, cv=skf, scoring="f1_macro", n_jobs=-1)
print("CV F1-macro (train): ", scores.mean(), "±", scores.std())
```

Dengan outputnya sebagai berikut:

```
CV F1-macro (train):  1.0 ± 0.0
```

Output menunjukkan bahwa **nilai rata-rata F1-macro pada validasi silang adalah 1.0**, menandakan model mampu mengklasifikasikan data training dengan sangat sempurna tanpa variasi performa antar fold.

5. **Langkah 5 - Tuning Ringkas (GridSearch)**

melakukan **tuning hyperparameter Random Forest** menggunakan **GridSearchCV** dengan validasi silang Stratified K-Fold (5 lipatan). Parameter yang dicoba adalah kombinasi dari max_depth dan min_samples_split. Proses ini mencari konfigurasi terbaik berdasarkan skor **F1-macro**. Setelah menemukan parameter optimal, model terbaik (best_rf) digunakan untuk memprediksi data validasi, lalu dievaluasi kembali dengan **F1-score** pada data tersebut.

```
# Langkah 4 - Tuning Ringkas (GridSearch)
from sklearn.model_selection import GridSearchCV

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

gs = GridSearchCV(pipe, 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_)
best_model = gs.best_estimator_
y_val_best = best_model.predict(X_val)
print("Best RF — F1(val):", f1_score(y_val, y_val_best, average="macro"))
```

dengan outputnya sebagai berikut:

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best params: {'clf__max_depth': None, 'clf__min_samples_split': 2}
Best RF — F1(val): 1.0
```

Output menunjukkan bahwa **GridSearchCV** menguji 12 kombinasi hyperparameter dengan 5-fold CV (total 60 training). Hasil terbaik diperoleh pada parameter max_depth=None dan min_samples_split=2. Model terbaik tersebut mencapai **F1-score = 1.0 pada data validasi**, yang berarti klasifikasi berjalan sempurna.

6. Langkah 6 - Evaluasi Akhir (Test Set)

Mengevaluasi **model terbaik (final_model)** pada data uji. Pertama, model memprediksi label (y_test_pred) lalu dihitung **F1-score**, **classification report**, dan **confusion matrix** untuk melihat akurasi tiap kelas. Jika model mendukung probabilitas (predict_proba), maka dihitung juga **ROC-AUC** dan dibuat grafik **ROC Curve** serta **Precision-Recall Curve**. Hasil grafik disimpan dalam file roc_test.png dan pr_test.png.

```python
# langkah 5 - Evaluasi Akhir (Test Set)
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, precision_recall_curve
import matplotlib.pyplot as plt

final_model = best_model  # pilih terbaik; jika baseline lebih baik, gunakan pipe

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 (bila 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)

    prec, rec, _ = precision_recall_curve(y_test, y_test_proba)
    plt.figure(); plt.plot(rec, prec); plt.xlabel("Recall"); plt.ylabel("Precision"); plt.title("PR Curve (test)")
    plt.tight_layout(); plt.savefig("pr_test.png", dpi=120)
```

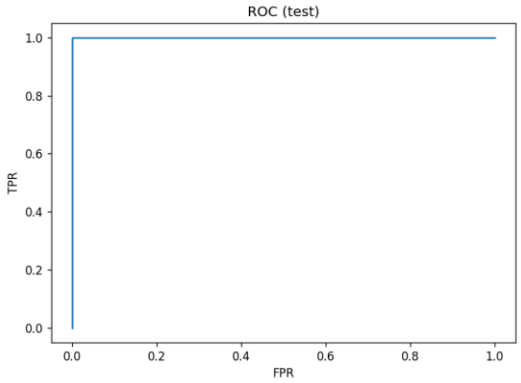
Outputnya:

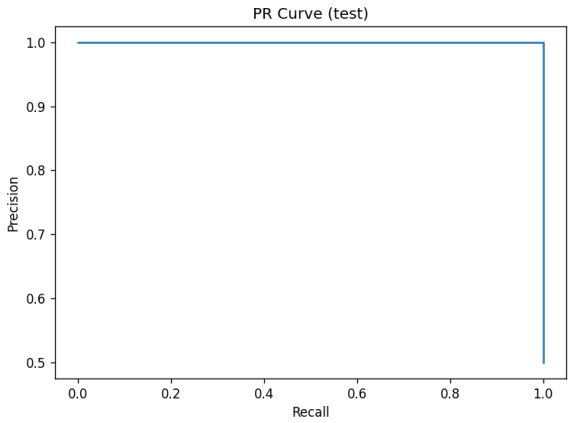
```
F1(test): 1.0
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         6
           1      1.000     1.000     1.000         6

    accuracy                         1.000        12
   macro avg      1.000     1.000     1.000        12
weighted avg      1.000     1.000     1.000        12

Confusion Matrix (test):
[[6 0]
 [0 6]]
ROC-AUC(test): 1.0
```





- F1(test) = 1.0 → model mampu menyeimbangkan precision dan recall secara maksimal.
- Classification report: untuk kedua kelas (0 dan 1), nilai precision, recall, dan f1-score = 1.0 dengan support masing-masing 6 data.
- Accuracy = 1.0 → semua 12 sampel test diprediksi benar.
- Confusion Matrix: [[6 0], [0 6]] menunjukkan tidak ada kesalahan prediksi (tidak ada False Positive maupun False Negative).
- ROC-AUC(test) = 1.0 → model memiliki kemampuan diskriminasi yang sempurna antara kelas positif dan negatif.

7. Langkah 7 – Pentingnya Fitur

- Bagian **6a (Feature importance native/Gini)** mengambil bobot pentingnya fitur langsung dari model Random Forest (`feature_importances_`), kemudian menampilkan 10 fitur teratas beserta nilainya.

- Bagian **6b (Permutation importance, opsional)** menyediakan alternatif evaluasi dengan cara mengacak nilai fitur lalu mengukur penurunan performa model, sehingga bisa menilai seberapa besar kontribusi masing-masing fitur terhadap prediksi.

```
# langkah 6 - pentingnya fitur
# 6a) Feature importance native (gini)
try:
    import numpy as np
    importances = final_model.named_steps["clf"].feature_importances_
    fn = final_model.named_steps["pre"].get_feature_names_out()
    top = sorted(zip(fn, importances), key=lambda x: x[1], reverse=True)
    print("Top feature importance:")
    for name, val in top[:10]:
        print(f"{name}: {val:.4f}")
except Exception as e:
    print("Feature importance tidak tersedia:", e)

# 6b) (Opsional) Permutation Importance
# from sklearn.inspection import permutation_importance
# r = permutation_importance(final_model, X_val, y_val, n_repeats=10, random_state=42, n_jobs=-1)
# ... (urutkan dan laporkan)
```

outputnya:

```
Top feature importance:
num__Rasio_Absensi: 0.2200
num__IPK: 0.2100
num__Jumlah_Absensi: 0.2000
num__Waktu_Belajar_Jam: 0.2000
num__IPK_x_Study: 0.1700
```

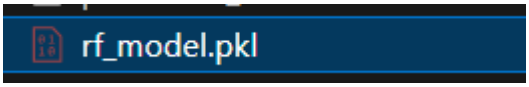
- Rasio Absensi (0.2200) → faktor paling dominan dalam menentukan hasil prediksi.
- IPK (0.2100) → hampir sama kuatnya pengaruhnya dengan rasio absensi.
- Jumlah Absensi (0.2000) dan Waktu Belajar (0.2000) → keduanya juga berperan besar.
- Interaksi IPK × Study (0.1700) → meskipun sedikit lebih kecil, tetap memberikan kontribusi penting.

8. **Langkah 8 – Simpan Model**

Dengan menggunakan joblib.dump, objek model (final_model) disimpan ke dalam file bernama **rf_model.pkl**. File ini bisa dimuat kembali nanti tanpa perlu melatih ulang model dari awal

```
# langkah 7 - Simpan Model
import joblib
joblib.dump(final_model, "rf_model.pkl")
print("Model disimpan sebagai rf_model.pkl")
```

Outputnya berupa file .pkl seperti pada gambar:



Output **rf_model.pkl** menunjukkan bahwa model Random Forest terbaik yang telah dilatih berhasil disimpan dalam sebuah file dengan format **pkl (pickle)**.

9. **Langkah 9 – Cek Inference Lokal**

- **Load model**: joblib.load("rf_model.pkl") memuat model Random Forest yang sebelumnya disimpan.
- **Buat data contoh**: sample adalah DataFrame berisi satu data fiktif dengan kolom yang sama seperti saat pelatihan model (IPK, Jumlah_Absensi, Waktu_Belajar_Jam, Rasio_Absensi, IPK_x_Study).
- **Prediksi**: mdl.predict(sample) menghitung output model untuk data tersebut. Hasil int(...) mengubah prediksi menjadi angka bulat.
- **Output**: Dicetak sebagai "Prediksi:", menampilkan apakah data contoh tersebut diprediksi **lulus (1)** atau **tidak lulus (0)**.

```
# Langkah 8 - Cek Inreference Lokal
# Contoh sekali jalan (input fiktif), sesuaikan nama kolom:
import pandas as pd, joblib
mdl = joblib.load("rf_model.pkl")
sample = pd.DataFrame([{
  "IPK": 3.4,
  "Jumlah_Absensi": 4,
  "Waktu_Belajar_Jam": 7,
  "Rasio_Absensi": 4/14,
  "IPK_x_Study": 3.4*7
}])
print("Prediksi:", int(mdl.predict(sample)[0]))
```

Outputnya:

Artinya, berdasarkan nilai:

- IPK = 3.4
- Jumlah Absensi = 4
- Waktu Belajar = 7 jam
- Rasio Absensi = 4/14 ≈ 0.286
- IPK × Waktu Belajar = 23.8

model menilai **mahasiswa ini memenuhi kriteria lulus/layak** menurut pola yang telah dipelajari dari data sebelumnya.