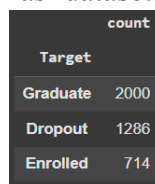


## Eksplorasi dan Prapemrosesan Data

Penelitian ini menggunakan data yang diambil dari *paper* berjudul “Predicting Student Dropout and Academic Success” (Valoriza et al. 2022). Dataset awal dimuat dan dilakukan pembersihan awal berupa penghapusan kolom non-informatif `sample_id`. Seluruh atribut pada dataset ini telah di-*encode* sehingga seluruh atribut bertipe data numerik. Di sisi lain, kolom target masih bertipe data *object*. Karena saya akan menguji beberapa model klasifikasi dan beberapa model klasifikasi tidak dapat menerima target bertipe data *object*, seperti XGBoost Classifier, maka saya melakukan *encoding* terhadap kolom target dengan menggunakan LabelEncoder dari library *sklearn.preprocessing*.

Setelah melakukan eksplorasi pada dataset, didapati fakta bahwa kolom target memiliki sebaran kelas yang tidak seimbang (*imbalanced class*) (lihat Gambar 1) sehingga diperlukan penanganan lebih lanjut untuk menyeimbangkan kelas supaya model tidak memiliki *bias* terhadap kelas yang mendominasi dataset.



Target	count
Graduate	2000
Dropout	1286
Enrolled	714

Gambar 1 Sebaran seluruh kelas dari dataset yang menunjukkan adanya permasalahan *imbalanced class*

Agar model dapat dilatih secara optimal, seluruh fitur numerik dinormalisasi menggunakan StandardScaler dari library *sklearn.preprocessing*. Normalisasi ini penting untuk menyetarakan skala fitur dan mempercepat konvergensi model, serta menghindari dominasi fitur tertentu dalam proses pembelajaran.

### Penyeimbangan Kelas (*Class Balancing*)

Dataset kemudian diatasi dari permasalahan ketidakseimbangan kelas (*imbalanced class*) menggunakan teknik SMOTE (Synthetic Minority Over-sampling Technique). SMOTE bekerja dengan membuat sampel sintetis baru dari kelas minoritas berdasarkan *nearest neighbors* dalam ruang fitur, sehingga distribusi kelas menjadi lebih seimbang dan model tidak cenderung "mengabaikan" kelas minoritas.

```
smt = SMOTE(random_state = 42)
X, y = smt.fit_resample(X, y)
```

### Pembagian Data

Setelah proses *oversampling*, dilakukan pembagian data menjadi data latih dan data uji dengan rasio pembagian 80% data latih dan 20% data uji. Pada proses ini digunakan argumen `stratify=y` yang berfungsi untuk memastikan bahwa proporsi kelas pada data pelatihan dan data pengujian tetap konsisten dengan distribusi kelas pada data asli. Ini penting untuk menjaga validitas evaluasi model, terutama ketika klasifikasi dilakukan pada kelas yang tidak seimbang.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
```

### Modeling

Dalam pengerjaan kompetisi ini, saya mengujikan model SVM, XGBoost, dan Random Forest. Ketiga model diterapkan *hyperparameter tuning* untuk mendapatkan parameter paling optimal berdasarkan karakteristik data. Namun, agar dapat memenuhi batasan halaman yang ditentukan, saya hanya akan membahas model random forest yang merupakan model pilihan final.

Parameter model dioptimasi menggunakan Randomized Search CV. Parameter yang dicari meliputi *n\_estimator* sebagai pengatur jumlah pohon dalam *forest*, *max\_depth* yang

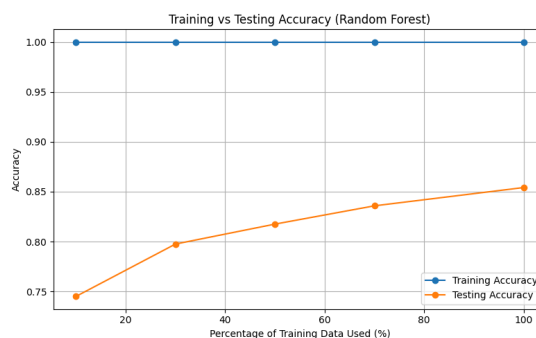
menentukan kedalaman maksimum pohon, *min\_samples\_split* dan *min\_samples\_leaf* yang mengontrol pemisahan *node*, *max\_features* yang mengatur jumlah fitur yang dipertimbangkan untuk pemisahan, serta *bootstrap* dan *class\_weight* yang melakukan pengaturan dan penyesuaian kelas. Daftar parameter yang digunakan dapat dilihat pada Gambar 2. Skor akurasi digunakan sebagai metrik evaluasi selama proses pencarian *hyperparameter*. Jumlah iterasi yang dilakukan adalah 50 iterasi dan proses pencarian dilakukan dengan *stratified k-fold cross-validation* dengan nilai *n\_splits* sebanyak 5 dan *n\_repeats* sebanyak 3.

```
param_dist_random_forest = {
    'n_estimators': np.arange(50, 250, 50), # 50, 100, 150, 200
    'max_depth': [None] + list(np.arange(5, 35, 5)), # None, 5, 10, 15, 20, 25, 30
    'min_samples_split': np.arange(2, 11, 2), # 2, 4, 6, 8, 10
    'min_samples_leaf': np.arange(1, 5), # 1, 2, 3, 4
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False],
    'class_weight': [None, 'balanced']
}
```

Gambar 2 Parameter yang digunakan dalam *randomized search cv random forest*. Diperoleh *best parameter* berupa *n\_estimators* = 50, *min\_samples\_split* = 4, *min\_samples\_leaf* = 1, *max\_features* = log2, *max\_depth* = None, *class\_weight* = None, dan *bootstrap* = False.

## Evaluasi

Diperoleh akurasi model pada data latih dan data uji secara berturut-turut sebesar 100% dan 85.25%. Untuk mengevaluasi pengaruh ukuran data terhadap kinerja model, dilakukan eksperimen dengan melatih model pada berbagai proporsi subset data pelatihan, mulai dari 10% hingga 100%. Akurasi hasil pelatihan dan pengujian dari masing-masing percobaan kemudian di-plot dalam grafik untuk menggambarkan perbandingan antara akurasi pelatihan dan akurasi pengujian (lihat Gambar 3). Hasil pengamatan terhadap grafik menunjukkan adanya kesenjangan yang cukup signifikan antara akurasi pelatihan dan akurasi pengujian, bahkan ketika persentase data pelatihan ditingkatkan. Temuan ini mengindikasikan bahwa model cenderung mengalami *overfitting*, yaitu kondisi di mana model sangat baik dalam mengenali pola pada data pelatihan namun gagal melakukan generalisasi terhadap data baru.



Gambar 3 Perbandingan akurasi training dan testing yang diujikan pada jumlah data yang berbeda-beda

Sebagai upaya mitigasi terhadap *overfitting*, dilakukan percobaan pelatihan ulang model dengan mengurangi jumlah fitur yang digunakan, yakni melatih model hanya menggunakan 20 dan 10 fitur teratas berdasarkan nilai *feature importance*. Namun, hasil evaluasi pada *leaderboard* Kaggle menunjukkan penurunan akurasi pada kedua pendekatan tersebut. Oleh karena itu, diputuskan untuk tetap menggunakan model awal tanpa pengurangan fitur. Model akhir kemudian digunakan untuk melakukan prediksi dan *submission* pada platform Kaggle, dengan akurasi akhir yang diperoleh sebesar 79.396%, skor terbesar dibandingkan pendekatan-pendekatan lainnya yang diujikan.

## KODE PROGRAM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score, RandomizedSearchCV
from sklearn import metrics
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_validate
from statistics import mean

from sklearn.ensemble import RandomForestClassifier
df_train = pd.read_csv('/content/drive/MyDrive/IPB/MATKUL/TINGKAT
3/SMT 6/Data Mining/KULIAH/Kaggle Competition/training.csv')

#drop kolom sample_id
df_train = df_train.drop(['sample_id'], axis=1)

#encode kolom Target
label_encoder = LabelEncoder()
df_train_encoded = df_train.copy()
df_train_encoded['Target'] =
label_encoder.fit_transform(df_train['Target'])
df_train_encoded

#Create Stratified K-fold cross validation
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)

#split data into 80 and 20
from sklearn.model_selection import train_test_split
X = df_train_encoded.drop(['Target'], axis=1)
y = df_train_encoded['Target']
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
smt = SMOTE(random_state = 42)
X, y = smt.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

#randomized search random forest
param_dist_random_forest = {
    'n_estimators': np.arange(50, 250, 50), # 50, 100, 150, 200
    'max_depth': [None] + list(np.arange(5, 35, 5)), # None, 5, 10,
15, 20, 25, 30
    'min_samples_split': np.arange(2, 11, 2), # 2, 4, 6, 8, 10
    'min_samples_leaf': np.arange(1, 5), # 1, 2, 3, 4
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False],
```

```

    'class_weight': [None, 'balanced']
}

rf = RandomForestClassifier(random_state = 42)

# RandomizedSearchCV
random_search_rf = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist_random_forest,
    n_iter=50, # Jumlah kombinasi parameter yang akan dicoba
    cv=cv, # Cross-validation
    scoring='accuracy',
    n_jobs=-1, # Gunakan semua core CPU
    verbose=1,
    random_state=42
)

random_search_rf.fit(X_train, y_train)

print(random_search_rf.best_params_) #{'n_estimators': np.int64(50),
'min_samples_split': np.int64(4), 'min_samples_leaf': np.int64(1),
'max_features': 'log2', 'max_depth': None, 'class_weight': None,
'bootstrap': False}

# ==== 5. Evaluasi pada Berbagai Ukuran Data ====
train_accuracies = []
val_accuracies = []
sizes = [0.1, 0.3, 0.5, 0.7, 1.0]

for size in sizes:
    subset_size = int(len(X_train) * size)
    X_sub = X_train[:subset_size]
    y_sub = y_train[:subset_size]

    model = RandomForestClassifier(n_estimators = 50,
min_samples_split = 4, min_samples_leaf = 1, max_features = 'log2',
max_depth = None, class_weight = None, bootstrap = False)
    model.fit(X_sub, y_sub)

    y_sub_pred = model.predict(X_sub)
    y_test_pred = model.predict(X_test)

    train_acc = accuracy_score(y_sub, y_sub_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    train_accuracies.append(train_acc)
    val_accuracies.append(test_acc)

# ==== 6. Grafik Akurasi ====
plt.figure(figsize=(8, 5))
plt.plot([int(s*100) for s in sizes], train_accuracies,
label='Training Accuracy', marker='o')
plt.plot([int(s*100) for s in sizes], val_accuracies, label='Testing
Accuracy', marker='o')
plt.xlabel('Percentage of Training Data Used (%)')
plt.ylabel('Accuracy')
plt.title('Training vs Testing Accuracy (Random Forest)')

```

```

plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ==== 7. Final Training dan Evaluasi ====
final_model_rf = RandomForestClassifier(n_estimators = 50,
                                       min_samples_split = 4,
                                       min_samples_leaf = 1,
                                       max_features = 'log2',
                                       max_depth = None,
                                       class_weight = None,
                                       bootstrap = False)

final_model_rf.fit(X_train, y_train)

# Akurasi Training dan Testing
train_acc_final = accuracy_score(y_train,
final_model_rf.predict(X_train))
test_acc_final = accuracy_score(y_test,
final_model_rf.predict(X_test))

print(f"🎯 Final Training Accuracy: {train_acc_final:.4f}")
print(f"🏠 Final Testing Accuracy: {test_acc_final:.4f}")

# feature importance

# Ambil nama-nama fitur dari X_train
feature_names = X_train.columns

# Ambil nilai feature importance
importances = final_model_rf.feature_importances_

# Gabungkan ke dalam DataFrame biar mudah dibaca dan diurutkan
feat_imp_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Tampilkan top-n fitur paling penting (misal top 20)
print("\nTop Feature Importances:")
#print(feat_imp_df.head(20))
print(feat_imp_df)

# Ambil top 20 fitur terpenting
top_20_features = feat_imp_df.head(20)['Feature'].tolist()

# Subset X_train dan X_test hanya dengan fitur-fitur tersebut
X_train_top20 = X_train[top_20_features]
X_test_top20 = X_test[top_20_features]

# latih ulang model
final_model_rf.fit(X_train_top20, y_train)

# Akurasi Training dan Testing
train_acc_final = accuracy_score(y_train,
final_model_rf.predict(X_train_top20))

```

```

test_acc_final = accuracy_score(y_test,
final_model_rf.predict(X_test_top20))

print(f"🎯 Final Training Accuracy: {train_acc_final:.4f}")
print(f"🏠 Final Testing Accuracy: {test_acc_final:.4f}")

# Ambil top 10 fitur terpenting
top_10_features = feat_imp_df.head(10)['Feature'].tolist()

# Subset X_train dan X_test hanya dengan fitur-fitur tersebut
X_train_top10 = X_train[top_10_features]
X_test_top10 = X_test[top_10_features]

# latih ulang model
final_model_rf.fit(X_train_top10, y_train)

# Akurasi Training dan Testing
train_acc_final = accuracy_score(y_train,
final_model_rf.predict(X_train_top10))
test_acc_final = accuracy_score(y_test,
final_model_rf.predict(X_test_top10))

print(f"🎯 Final Training Accuracy: {train_acc_final:.4f}")
print(f"🏠 Final Testing Accuracy: {test_acc_final:.4f}")

df_test = pd.read_csv('/content/drive/MyDrive/IPB/MATKUL/TINGKAT
3/SMT 6/Data Mining/KULIAH/Kaggle Competition/testing.csv')

df_test_selected = df_test.drop(['sample_id'], axis=1)

df_test_selected_scaled = scaler.transform(df_test_selected)

df_test_selected = pd.DataFrame(scaler.transform(df_test_selected),
columns=df_test_selected.columns)

y_pred = final_model_rf.predict(df_test_selected)

y_pred_labels = label_encoder.inverse_transform(y_pred)

df_prediksi = pd.DataFrame({
    "sample_id": df_test["sample_id"],
    "Target": y_pred_labels
})

df_prediksi.to_csv("hasil_prediksi.csv", index=False)

```

Catatan: program ini tidak memuat secara lengkap keseluruhan program dari pendekatan yang dilakukan, melainkan hanya memuat program dari pendekatan yang dijelaskan pada halaman 1-2. Untuk lebih lengkap, silakan merujuk kepada Python notebook yang dilampirkan bersamaan dengan file ini di dalam ZIP.