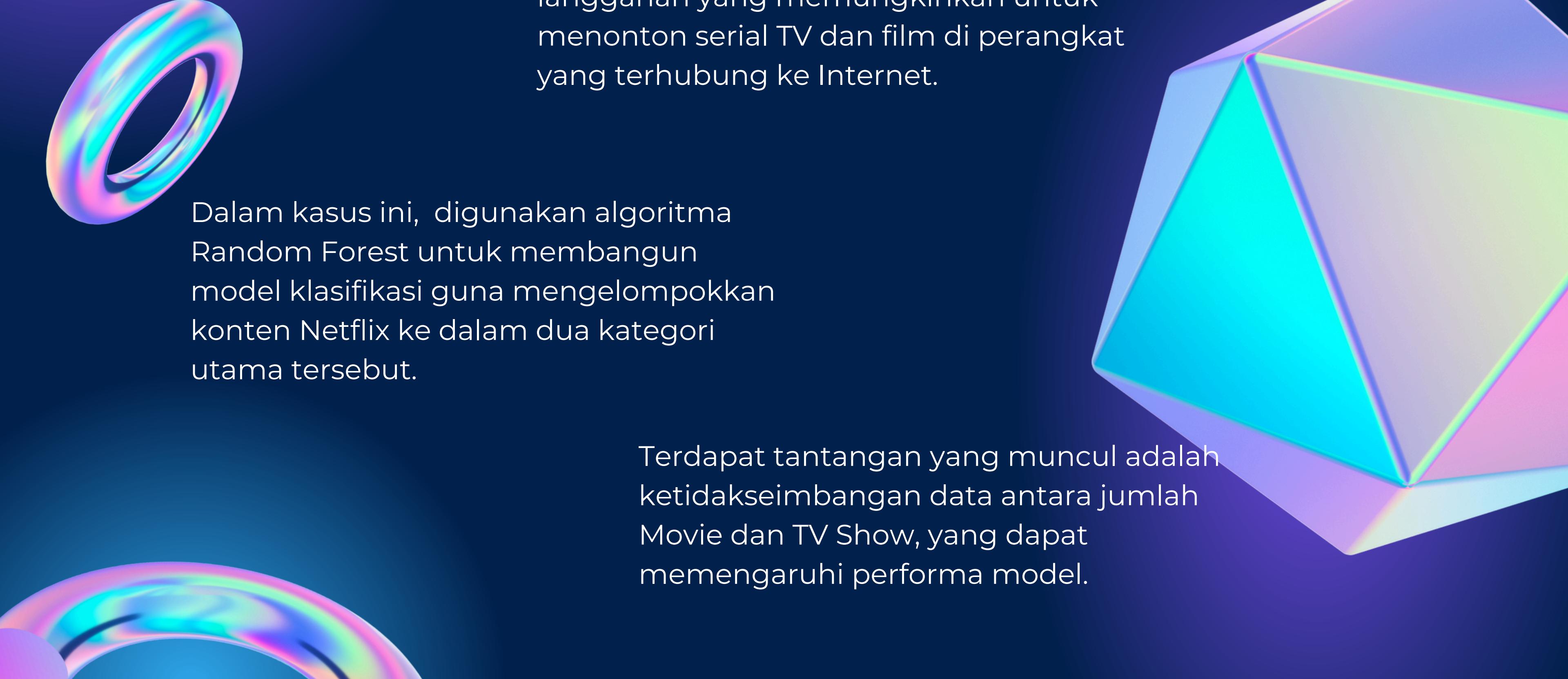


Implementasi Random Forest Untuk Klasifikasi Acara Tv dan Movie di Netflix

Osama Alfa

Latar Belakang

The background features abstract, semi-transparent 3D geometric shapes in shades of blue, purple, and green, floating against a dark blue gradient background.

Netflix adalah layanan streaming berbasis langganan yang memungkinkan untuk menonton serial TV dan film di perangkat yang terhubung ke Internet.

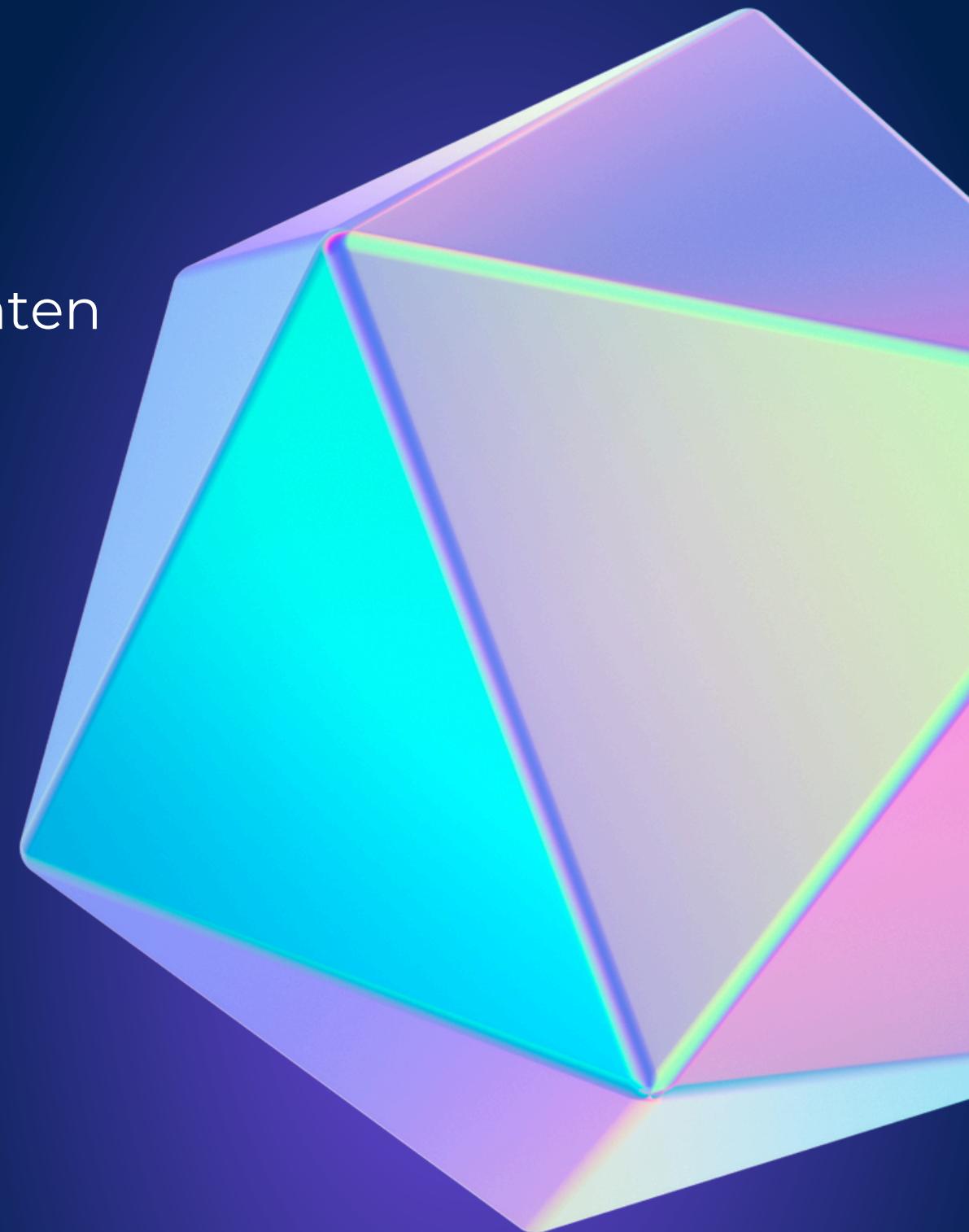
Dalam kasus ini, digunakan algoritma Random Forest untuk membangun model klasifikasi guna mengelompokkan konten Netflix ke dalam dua kategori utama tersebut.

Terdapat tantangan yang muncul adalah ketidakseimbangan data antara jumlah Movie dan TV Show, yang dapat memengaruhi performa model.

Tujuan

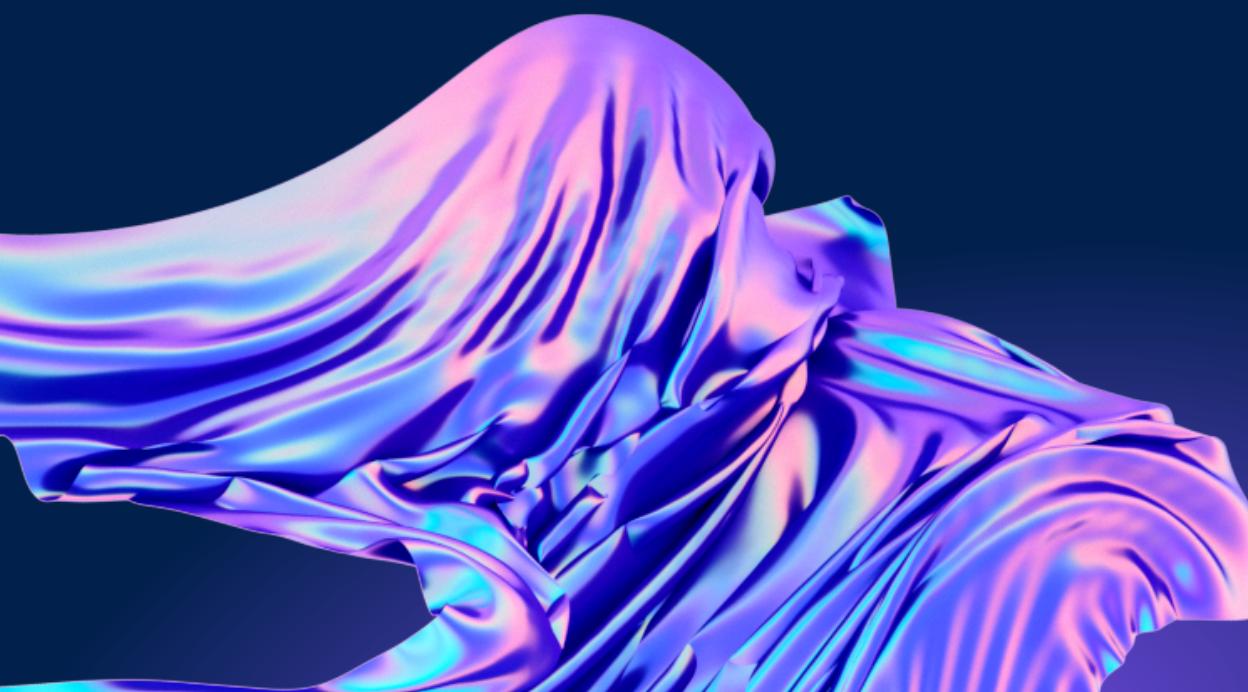
Menggunakan algoritma Random Forest dengan teknik perbaikan seperti SMOTE (Synthetic Minority Over-sampling Technique), optimasi parameter dengan Grid Search, serta penyesuaian threshold prediksi guna meningkatkan akurasi dan keseimbangan model.

Membangun Model Machine Learning untuk klasifikasi pengelompokkan konten Netflix ke dalam dua kategori utama tersebut.



Dataset

Dataset yang digunakan berasal dari Kaggle yang berisi Daftar Film yang ada di Netflix yang memiliki lebih dari 8000 acara movie / tv diplatform tersebut. Kumpulan data tabel ini terdiri dari daftar semua film dan acara TV yang tersedia di Netflix, beserta detail seperti - pemeran, sutradara, peringkat, tahun rilis, durasi, dan lainnya.



Load Dataset

- Memuat Dataset dari Kaggle
- Melihat Kolom-kolom yang ada di dalam dataset

```
▶ # Memasukkan Library yang akan Digunakan
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Memasukkan Dataset yang akan Diolah
df = pd.read_csv('/content/drive/MyDrive/Data Science/Data Analytics/Netflix Movies and TV Show Analysis/netflix_titles.csv')

# Menampilkan Data yang ada di Baris Atas
df.head()
```

Cleaning Data

- Memeriksa dan menghapus data null dan duplikat.
- Mengkonversi kolom penanggalan ke format datetime

```
[32] # Mengatasi nilai yang hilang dengan mengganti NaN dengan "Unknown"
df.fillna({"director": "Unknown", "cast": "Unknown", "country": "Unknown",
           "date_added": "Unknown", "rating": "Unknown",
           "duration": "Unknown"}, inplace=True)

# Mengonversi 'date_added' ke format datetime, mengganti 'Unknown' dengan NaT
df["date_added"] = pd.to_datetime(df["date_added"], errors="coerce")

# Menampilkan jumlah nilai yang hilang setelah pembersihan
df.isnull().sum()
```

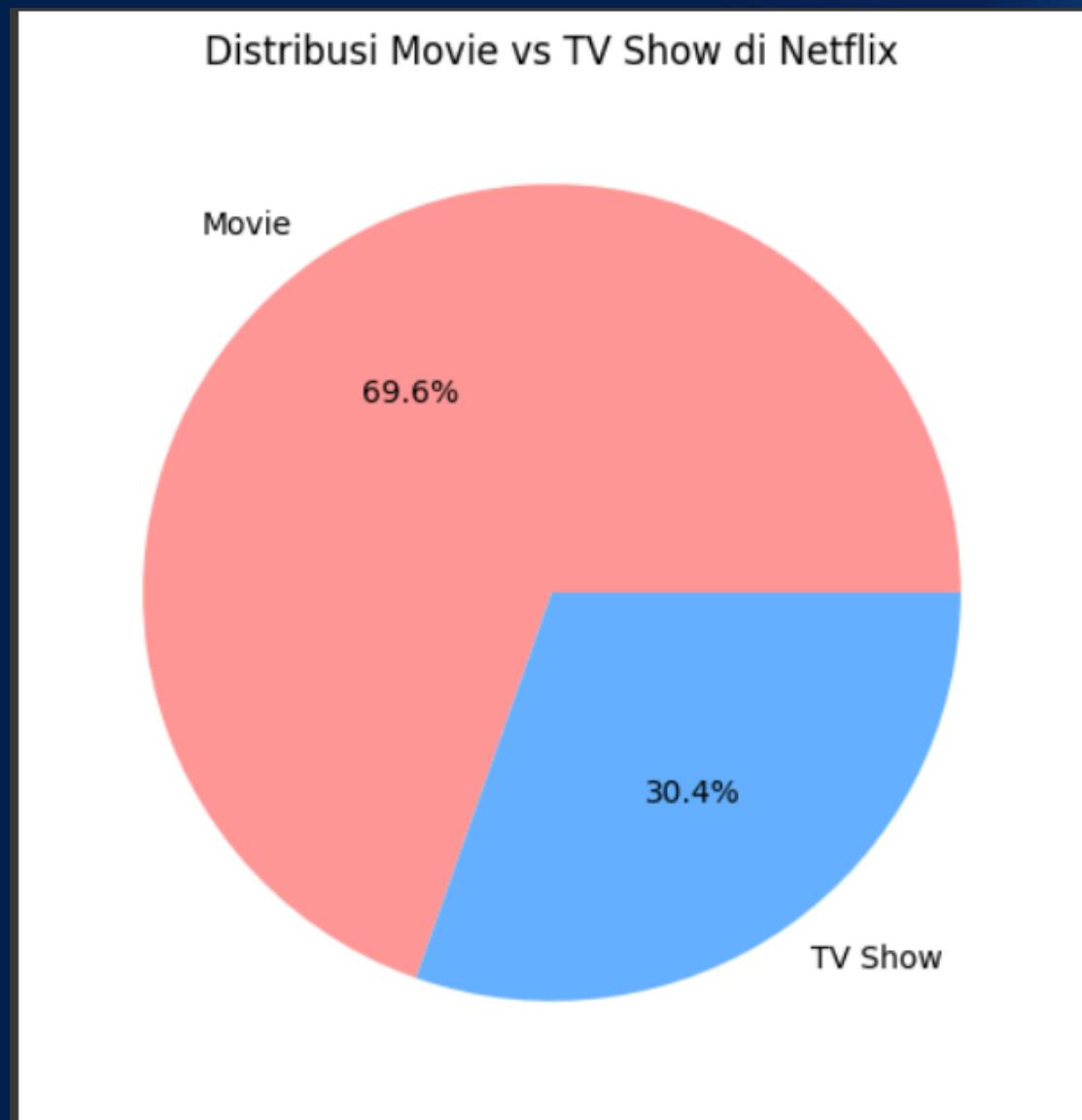
Exploratory Data Analysis

Kita akan membuat pie chart untuk menampilkan persentase dari jumlah Movie dan TV Show yang ada di Netflix.

```
# Memasukkan Library yang Diperlukan untuk Visualisasi Data
import matplotlib.pyplot as plt
import seaborn as sns

# Menghitung jumlah Movie dan TV Show
type_counts = df["type"].value_counts()

# Membuat pie chart
plt.figure(figsize=(6, 6))
plt.pie(type_counts, labels=type_counts.index, autopct='%1.1f%%',
        colors=["#ff9999", "#66b3ff"])
plt.title("Distribusi Movie vs TV Show di Netflix")
plt.show()
```



Exploratory Data Analysis

Lalu membuat Work Cloud untuk memvisualisasikan data teks secara visual. Work Cloud ini berisi tentang deskripsi konten yang ada pada Netflix.

```
# Memasukkan Library untuk Membuat WordCloud
from wordcloud import WordCloud

# Menggabungkan semua deskripsi menjadi satu teks besar
text = " ".join(df["description"].dropna())

# Membuat WordCloud
wordcloud = WordCloud(width=800, height=400, background_color="black",
                      colormap="cool").generate(text)

# Menampilkan WordCloud
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud Deskripsi Konten Netflix")
plt.show()
```



Preprocessing Data

Tahap selanjutnya kita akan membagi target untuk melakukan pelatihan data. Di sini kita akan menggunakan fitur teks 'description' untuk klasifikasi.

```
# Memasukkan Library Penting Untuk Modelling
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer

# Menggunakan fitur teks 'description' untuk klasifikasi
df_model = df[["description", "type"]].copy()
|
# Encoding label ('Movie' = 0, 'TV Show' = 1)
label_encoder = LabelEncoder()
df_model["type"] = label_encoder.fit_transform(df_model["type"])

# TF-IDF Vectorizer untuk mengubah teks menjadi fitur numerik
tfidf = TfidfVectorizer(stop_words="english", max_features=5000)
X = tfidf.fit_transform(df_model["description"])
y = df_model["type"]

# Membagi data menjadi training (80%) dan testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)

# Menampilkan bentuk dataset setelah preprocessing
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Modeling

Kemudian, kita akan melakukan modelling dengan menggunakan Random Forest Classifier.

```
# Memasukkan Library untuk Modelling
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier

# Melatih model Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(x_train, y_train)

# Memprediksi hasil pada data uji
y_pred_rf = rf_model.predict(x_test)

# Evaluasi model Random Forest
accuracy_rf = accuracy_score(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Menampilkan hasil evaluasi secara rapi
print("== Evaluasi Model Random Forest ==")
print(f"Akurasi: {accuracy_rf:.4f}\n")
print("Laporan Klasifikasi:")
print(report_rf)
print("Confusion Matrix:")
print(conf_matrix_rf)
```

== Evaluasi Model Random Forest ==
Akurasi: 0.7429

Laporan Klasifikasi:

	precision	recall	f1-score	support
0	0.75	0.94	0.83	1214
1	0.69	0.31	0.43	548
accuracy			0.74	1762
macro avg	0.72	0.62	0.63	1762
weighted avg	0.73	0.74	0.71	1762

Confusion Matrix:

```
[[1139  75]
 [ 378 170]]
```

Evaluasi

Dalam tahap ini , kita akan menggunakan SMOTE untuk menyeimbangkan kelas.

```
# Menerapkan SMOTE untuk menyeimbangkan kelas
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Membagi ulang data setelah oversampling
X_train_res, X_test_res, y_train_res, y_test_res = train_test_split(
    X_resampled, y_resampled, test_size=0.2, random_state=42)

# Melatih ulang model Random Forest
rf_model_res = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model_res.fit(X_train_res, y_train_res)

# Memprediksi hasil pada data uji setelah SMOTE
y_pred_res = rf_model_res.predict(X_test_res)
```

Evaluasi

Selanjutnya, kita akan melakukan evaluasi model dengan menggunakan Grid Search untuk menemukan kombinasi parameter terbaik

```
# Menentukan Grid Search untuk Mencari Parameter Terbaik
from sklearn.model_selection import GridSearchCV

# Memasukkan Parameter
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf, param_grid, cv=3, n_jobs=-1, scoring='accuracy')
grid_search.fit(x_train, y_train)

# Model terbaik
best_rf = grid_search.best_estimator_
y_pred_proba = best_rf.predict_proba(x_test)[:, 1] # Probabilitas kelas 1 (TV Show)
```

Evaluasi

Lalu kita akan menyesuaikan Threshold Prediksi dengan menggunakan predict_proba untuk mendapatkan probabilitas setiap prediksi.

```
# Menyesuaikan Threshold Prediksi
import numpy as np
thresholds = np.arange(0.3, 0.7, 0.05)
best_threshold = 0.5
best_f1 = 0

for threshold in thresholds:
    y_pred_adjusted = (y_pred_proba >= threshold).astype(int)
    f1 = classification_report(y_test, y_pred_adjusted, output_dict=True)[
        'weighted avg']['f1-score']
    if f1 > best_f1:
        best_f1 = f1
        best_threshold = threshold
```

Evaluasi

Dan yang terakhir, melakukan evaluasi model setelah optimasi.

```
# Evaluasi Model Setelah Optimasi
y_pred_final = (y_pred_proba >= best_threshold).astype(int)
accuracy_final = accuracy_score(y_test, y_pred_final)
report_final = classification_report(y_test, y_pred_final)
conf_matrix_final = confusion_matrix(y_test, y_pred_final)

# Menampilkan hasil evaluasi
print("== Evaluasi Model Random Forest setelah Optimasi ==")
print(f"Akurasi: {accuracy_final:.4f}")
print(f"Best Threshold: {best_threshold}")
print("Laporan Klasifikasi:")
print(report_final)
print("Confusion Matrix:")
print(conf_matrix_final)
```

Evaluasi

Sebelum Optimasi

```
== Evaluasi Model Random Forest ==
Akurasi: 0.7429

Laporan Klasifikasi:
precision    recall    f1-score   support
0            0.75     0.94      0.83     1214
1            0.69     0.31      0.43      548

accuracy          0.74     1762
macro avg       0.72     0.62      0.63     1762
weighted avg    0.73     0.74      0.71     1762

Confusion Matrix:
[[1139  75]
 [ 378 170]]
```

Setelah Optimasi

```
== Evaluasi Model Random Forest setelah Optimasi ==
Akurasi: 0.7452
Best Threshold: 0.35
Laporan Klasifikasi:
precision    recall    f1-score   support
0            0.75     0.95      0.84     1214
1            0.73     0.29      0.41      548

accuracy          0.75     1762
macro avg       0.74     0.62      0.63     1762
weighted avg    0.74     0.75      0.71     1762

Confusion Matrix:
[[1154  60]
 [ 389 159]]
```

Akurasi

- Akurasi: 74.52%
- Model dapat memprediksi dengan benar sebanyak 74.52% dari total data uji.
- Ini merupakan hasil setelah menerapkan SMOTE, Grid Search, dan penyesuaian threshold.

Recall

- Movie (0) = 95% → Model sangat baik dalam mengenali Movie, hampir semua Movie terdeteksi.
- TV Show (1) = 29% → Model hanya mengenali 29% dari semua TV Show yang benar-benar ada!
- Recall TV Show sangat rendah, berarti banyak TV Show salah diklasifikasikan sebagai Movie.

Hasil

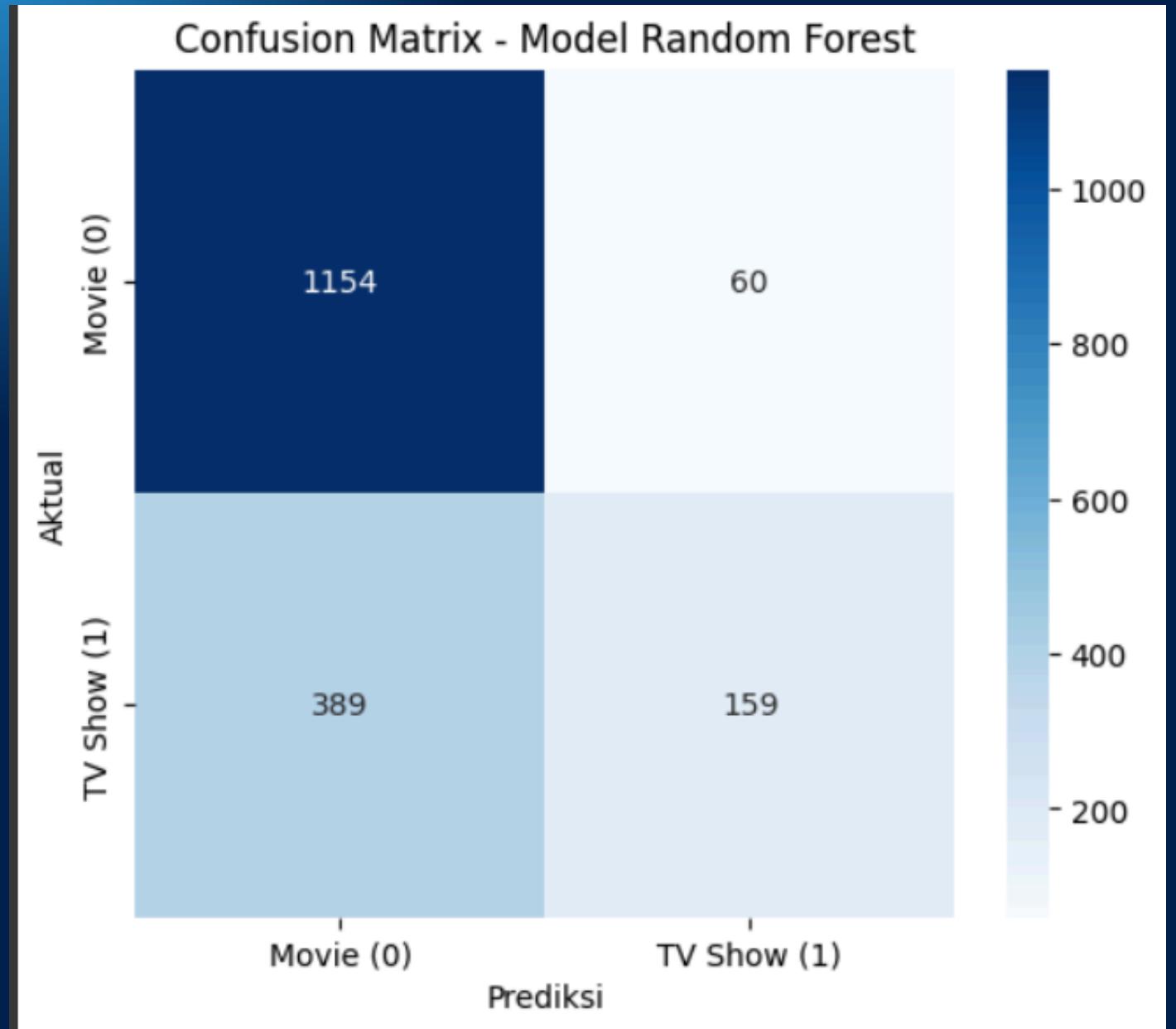
Presisi

- Movie (0) = 75% → Dari semua yang diprediksi sebagai Movie, 75% benar.
- TV Show (1) = 73% → Dari semua yang diprediksi sebagai TV Show, 73% benar.
- Precision seimbang antara kedua kelas.

F1-Score

- Movie (0) = 0.84, TV Show (1) = 0.41 → Kinerja model lebih baik pada Movie.
- TV Show sulit diklasifikasikan dengan benar, karena recall rendah.

Confusion Matrix



Interpretasi Confusion Matrix:

- 1154 Movie diklasifikasikan dengan benar sebagai Movie
- 159 TV Show diklasifikasikan dengan benar sebagai TV Show
- 60 Movie salah diklasifikasikan sebagai TV Show
- 389 TV Show salah diklasifikasikan sebagai Movie

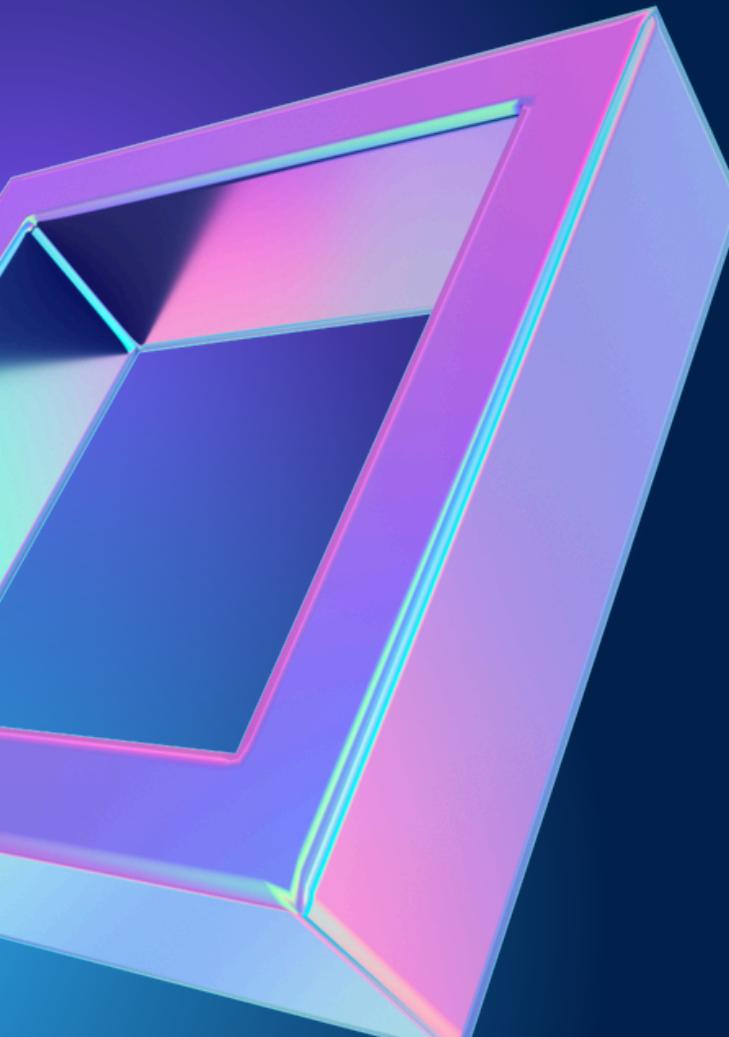
Kesimpulan

Penerapan SMOTE berhasil meningkatkan performa model secara signifikan, terutama dalam mengenali TV Show yang sebelumnya sulit diklasifikasikan.

Grid Search meningkatkan performa model dengan parameter optimal.

Penyesuaian threshold prediksi membantu meningkatkan F1-score dan keseimbangan klasifikasi.

Namun, masih ada masalah di model yang lebih cenderung mengenali Movie dengan baik tetapi kesulitan dengan TV Show. Karena itu, Visualisasi Confusion Matrix membantu memahami pola kesalahan model.



Thank You