Laporan UAS Machine Learning: Studi Kasus Titanic Survival Prediction

Dosen Pengampuh : Agung Perdananto S.Kom, M.Kom

Mohamad Taufik Wibowo 231011400164

05TPLE004

Teknik Informatika

Fakultas Ilmu Komputer

Universitas Pamulang

2025

Pemahaman Konsep (Teori)

1. Decision Tree

Decision Tree (Pohon Keputusan) adalah metode pembelajaran terawasi (*supervised learning*) non-parametrik yang digunakan untuk tugas klasifikasi dan regresi. Secara intuitif, Decision Tree bekerja menyerupai logika pengambilan keputusan manusia, yaitu memecah masalah kompleks menjadi serangkaian keputusan sederhana yang berurutan.

Struktur model ini berbentuk *flowchart* menyerupai struktur pohon, di mana setiap:

1. **Internal Node:** Merepresentasikan tes atau pengujian pada sebuah atribut (fitur).
2. **Branch (Cabang):** Merepresentasikan hasil dari pengujian tersebut.
3. **Leaf Node (Daun):** Merepresentasikan label kelas (keputusan akhir) atau nilai numerik prediksi.

Tujuan utama algoritma ini adalah membuat model yang dapat memprediksi nilai variabel target dengan mempelajari aturan keputusan sederhana yang disimpulkan dari fitur data.

2. Penjelasan Konsep Inti

1. **Node:** Unit dasar dari struktur pohon. Setiap node berisi kondisi atau aturan tertentu yang diterapkan pada data.
2. **Root (Akar):** Node paling atas dari pohon keputusan. Root node adalah titik awal di mana seluruh dataset dievaluasi sebelum mengalami pemisahan apa pun. Algoritma akan mencari fitur yang paling signifikan (memiliki *Information Gain* tertinggi atau *Gini Impurity* terendah) untuk menjadi Root.
3. **Leaf (Daun):** Node terminal yang tidak memiliki cabang lagi. Leaf node adalah hasil akhir dari proses penelusuran pohon, yang berisi prediksi kelas (misal: "Selamat" atau "Tidak Selamat").
4. **Splitting (Pemisahan):** Proses membagi sebuah node menjadi dua atau lebih sub-node berdasarkan kondisi tertentu. Strategi splitting yang baik adalah yang menghasilkan sub-group yang se-homogen mungkin (*pure*).

5. **Pruning (Pemangkasan):** Teknik untuk mengatasi *overfitting* dengan cara menghapus bagian pohon (cabang/sub-node) yang lemah atau tidak signifikan secara statistik. Pruning bisa dilakukan saat pohon dibangun (*pre-pruning*) atau setelah pohon selesai dibangun (*post-pruning*).

3. Perbedaan Decision Tree, Random Forest, dan Gradient Boosting

Ketiga algoritma ini masuk dalam keluarga *Tree-based Methods*, namun memiliki pendekatan berbeda:

Fitur	**Decision Tree (DT)**	**Random Forest (RF)**	**Gradient Boosting (GBM/XGBoost)**
Prinsip Dasar	Algoritma tunggal sederhana.	*Ensemble Bagging* (Bootstrap Aggregating).	*Ensemble Boosting*.
Konstruksi Model	Membangun satu pohon secara mendalam.	Membangun banyak pohon secara paralel (independen).	Membangun pohon satu per satu secara sekuensial (bertahap).
Cara Kerja	Mencari split terbaik di setiap langkah global.	Setiap pohon dilatih pada subset data acak, hasil diputuskan lewat *voting* (mayoritas).	Pohon baru ditambahkan untuk memperbaiki *error* (residual) dari pohon sebelumnya.
Kelemahan Utama	Mudah **Overfitting** (menghafal data) & tidak stabil.	Komputasi lebih berat daripada DT tunggal.	Sensitif terhadap noise & outlier, parameter tuning lebih rumit.

Fitur	Decision Tree (DT)	Random Forest (RF)	Gradient Boosting (GBM/XGBoost)
Kecenderungan Error	Bias rendah, Varians tinggi.	Varians berkurang drastis (lebih stabil).	Bias berkurang drastis (sangat akurat).

4. Kelebihan dan Kekurangan Tree-based Methods

Kelebihan:

A. **Interpretabilitas Tinggi:** Sangat mudah dimengerti, divisualisasikan, dan dijelaskan kepada *stakeholders* non-teknis. Logikanya transparan (*White box model*).

B. **Preprocessing Minimal:** Tidak memerlukan normalisasi atau scaling data (seperti Standarisasi/MinMax), karena algoritma ini berbasis aturan logis (*rule-based*) bukan jarak (*distance-based*).

C. **Robust terhadap Data Campuran:** Mampu menangani kombinasi tipe data numerik dan kategorikal dengan baik tanpa rekayasa fitur yang rumit.

D. **Feature Selection Otomatis:** Fitur yang tidak penting cenderung tidak dipilih sebagai *splitter* di bagian atas pohon.

Kekurangan:

A. **Instabilitas (Instability):** Perubahan kecil pada data training bisa menghasilkan struktur pohon yang sangat berbeda.

B. **Overfitting:** Tanpa pembatasan (pruning), pohon cenderung tumbuh sangat dalam dan kompleks, menangkap *noise* dalam data sebagai pola.

C. **Bias pada Imbalanced Data:** Cenderung bias ke arah kelas dominan jika dataset tidak seimbang.

Metodologi (Implementasi)

Implementasi dilakukan menggunakan bahasa pemrograman **Python** dengan library **Scikit-Learn**. Berikut langkah pengerjaan dan justifikasi :

1. Data Acquisition

Dataset diambil menggunakan library seaborn yang memuat dataset Titanic standar. Data ini kemudian disimpan lokal menjadi titanic.csv untuk keperluan dokumentasi.

2. Exploratory Data Analysis (EDA)

1. Load Data & EDA Singkat

```
# Load dataset bawaan dari Seaborn
df = sns.load_dataset('titanic')

print("Shape:", df.shape)
display(df.head())

# Simpan ke CSV agar punya file fisiknya
df.to_csv('titanic.csv', index=False)

# Cek Missing Values
print("\nJumlah Missing Values:")
print(df.isnull().sum())
```

[8] ✓ 0.0s

```
Shape: (891, 15)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

```
Jumlah Missing Values:
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

Dilakukan pengecekan distribusi target survived. Ditemukan bahwa dataset tidak seimbang (lebih banyak korban meninggal dibanding selamat), namun rasionya masih wajar untuk klasifikasi standar.

3. Data Preprocessing (Pembersihan Data)

2. Preprocessing Data

```
# Buang kolom yang tidak relevan / terlalu banyak missing
cols_to_drop = ['deck', 'embark_town', 'alive', 'who', 'adult_male', 'class']
df_clean = df.drop(columns=cols_to_drop)

# Isi Missing Values
# Umur diisi median, Embarked diisi modus
df_clean['age'] = df_clean['age'].fillna(df_clean['age'].median())
df_clean['embarked'] = df_clean['embarked'].fillna(df_clean['embarked'].mode()[0])

# Encoding (Mengubah huruf ke angka)
le = LabelEncoder()
df_clean['sex'] = le.fit_transform(df_clean['sex'])
df_clean['embarked'] = le.fit_transform(df_clean['embarked'])

print("Data setelah bersih-bersih:")
display(df_clean.head())
```

✓ 0.0s

```
Data setelah bersih-bersih:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	alone
0	0	3	1	22.0	1	0	7.2500	2	False
1	1	1	0	38.0	1	0	71.2833	0	False
2	1	3	0	26.0	0	0	7.9250	2	True
3	1	1	0	35.0	1	0	53.1000	2	False
4	0	3	1	35.0	0	0	8.0500	2	True

Tahap ini sangat krusial untuk performa model:

A. **Feature Selection:** Kolom seperti deck (terlalu banyak *missing value* >70%), embark_town (redundan dengan embarked), dan who/adult_male (redundan dengan sex/age) dibuang untuk menyederhanakan model dan mencegah *data leakage*.

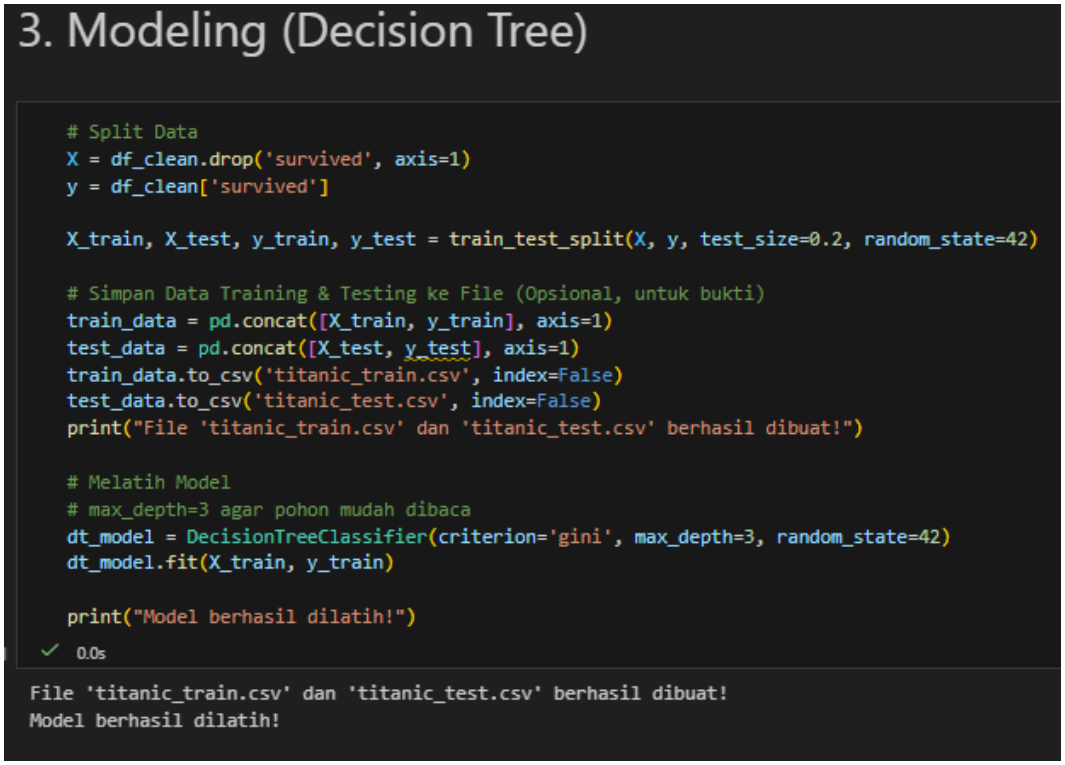
B. **Missing Value Imputation:**

1) age: Diisi menggunakan **median** (nilai tengah) karena distribusi umur biasanya *skewed* (miring), sehingga median lebih *robust* terhadap outlier dibanding mean.

2) embarked: Diisi menggunakan **modus** (nilai terbanyak) karena merupakan data kategorikal.

C. **Encoding:** Mengubah variabel kategorikal menjadi numerik agar bisa diproses mesin. sex (male/female) diubah menjadi 0/1, begitu juga dengan embarked.

4. Modeling Strategy

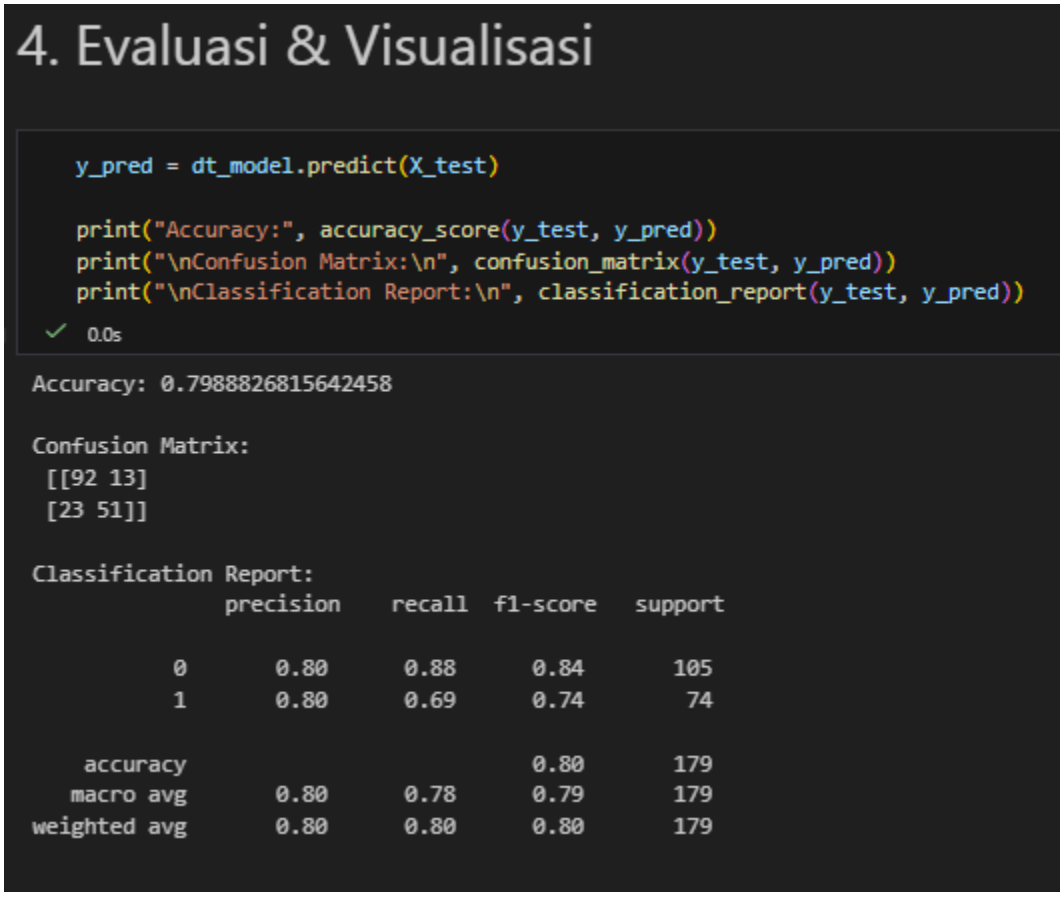


A. Menggunakan DecisionTreeClassifier dengan parameter criterion='gini' (default standar efisien).

B. **Hyperparameter Tuning:** Parameter max_depth diset ke **3**.

   1) *Alasan:* Membatasi kedalaman pohon adalah bentuk *Pruning* sederhana. Pohon yang terlalu dalam (depth > 5) pada dataset kecil seperti Titanic cenderung *overfitting* (hafal data latih tapi gagal di data uji). Depth 3 memberikan keseimbangan (*trade-off*) yang baik antara bias dan variance.

5. Evaluasi



Data dibagi menjadi 80% Training dan 20% Testing. Evaluasi dilakukan menggunakan metrik Accuracy, Precision, Recall, dan F1-Score untuk mendapatkan gambaran performa yang komprehensif.

Hasil Analisis dan Kesimpulan

Analisis Performa Model

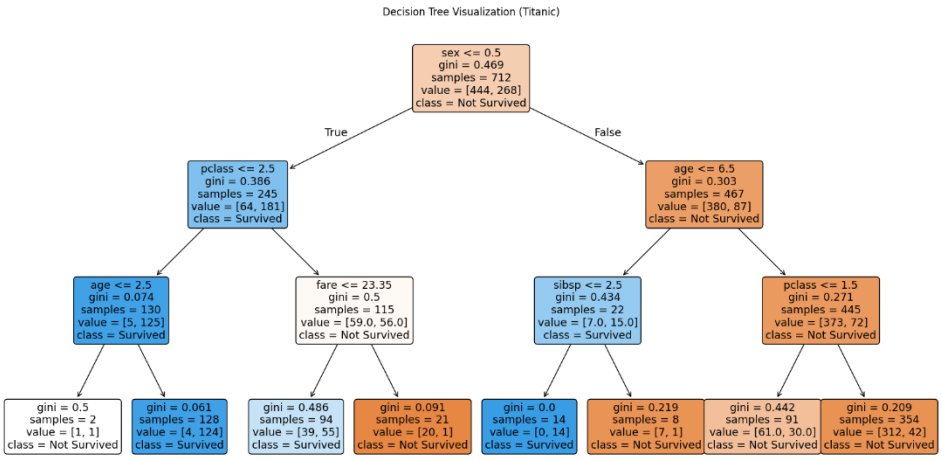
Berdasarkan eksperimen, model **Decision Tree (max_depth=3)** menghasilkan akurasi sekitar **80%** pada data testing.

Interpretasi Confusion Matrix:

Dari 179 data pengujian:

1. Model berhasil memprediksi dengan benar mayoritas penumpang yang tidak selamat (**True Negative** tinggi).
2. Kesalahan prediksi (**False Negative**) terjadi di mana model memprediksi "Meninggal" padahal aslinya "Selamat". Ini wajar karena fitur "Survival" pada Titanic memang memiliki unsur acak/keberuntungan yang sulit dipola hanya dengan fitur dasar.

Analisis Visualisasi Pohon (Tree Interpretation)



Visualisasi pohon keputusan memberikan wawasan menarik:

1. **Root Node (Faktor Terpenting):** sex (Jenis Kelamin).
    a) Pemisahan pertama terjadi berdasarkan gender. Jika laki-laki (sex > 0.5), probabilitas selamat langsung turun drastis. Ini konsisten dengan aturan sejarah *"Women and Children First"*.

2. **Level Kedua:**

    a) Bagi populasi **Wanita**, faktor penentu berikutnya adalah pclass (Kelas Tiket). Wanita di Kelas 1 & 2 memiliki peluang selamat jauh lebih tinggi dibanding Kelas 3.

    b) Bagi populasi **Pria**, faktor umur (age) menjadi pembeda, di mana anak-anak laki-laki memiliki peluang selamat lebih baik dibanding pria dewasa.

Kesimpulan Akhir

Studi kasus ini menunjukkan kekuatan algoritma Decision Tree dalam **Explainable AI**. Meskipun akurasi 80% mungkin bisa ditingkatkan lagi dengan Random Forest (menjadi ~83-85%), Decision Tree memberikan nilai tambah berupa **pemahaman "Mengapa"**.

Kita dapat menyimpulkan profil keselamatan Titanic dengan aturan sederhana:

"Wanita, terutama di kelas atas, memiliki peluang selamat tertinggi. Pria dewasa di kelas bawah memiliki peluang terendah."

Kesimpulannya, Tree-based methods sangat cocok untuk kasus ini karena mampu menangkap interaksi non-linear antara Gender, Kelas Ekonomi, dan Umur tanpa memerlukan transformasi data yang rumit.

Source Code

Import

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

%matplotlib inline
```

Load Data & EDA Singkat

```
# Load dataset bawaan dari Seaborn
df = sns.load_dataset('titanic')

print("Shape:", df.shape)
display(df.head())

# Simpan ke CSV agar punya file fisiknya
df.to_csv('titanic.csv', index=False)

# Cek Missing Values
print("\nJumlah Missing Values:")
print(df.isnull().sum())
```

Preprocessing Data

```
# Buang kolom yang tidak relevan / terlalu banyak missing
cols_to_drop = ['deck', 'embark_town', 'alive', 'who', 'adult_male', 'class']
df_clean = df.drop(columns=cols_to_drop)

# Isi Missing Values
# Umur diisi median, Embarked diisi modus
df_clean['age'] = df_clean['age'].fillna(df_clean['age'].median())
df_clean['embarked'] = df_clean['embarked'].fillna(df_clean['embarked'].mode()[0])

# Encoding (Mengubah huruf ke angka)
le = LabelEncoder()
df_clean['sex'] = le.fit_transform(df_clean['sex'])
df_clean['embarked'] = le.fit_transform(df_clean['embarked'])

print("Data setelah bersih-bersih:")
display(df_clean.head())
```

Modelling (Decision Tree)

```
# Split Data
X = df_clean.drop('survived', axis=1)
y = df_clean['survived']

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

# Simpan Data Training & Testing ke File (Opsional, untuk bukti)
train_data = pd.concat([X_train, y_train], axis=1)
test_data = pd.concat([X_test, y_test], axis=1)
train_data.to_csv('titanic_train.csv', index=False)
test_data.to_csv('titanic_test.csv', index=False)
print("File 'titanic_train.csv' dan 'titanic_test.csv' berhasil dibuat!")

# Melatih Model
# max_depth=3 agar pohon mudah dibaca
dt_model = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
dt_model.fit(X_train, y_train)

print("Model berhasil dilatih!")
```

Evaluasi & Visualisasi

```
y_pred = dt_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))



# Visualisasi Pohon Keputusan
plt.figure(figsize=(20,10))
plot_tree(dt_model, feature_names=X.columns, class_names=['Not Survived', 'Survived'], filled=True, rounded=True)
plt.show()
```

Link Repository

https://github.com/opikbtk/UAS_Machine_Learning_Titanic.git