Tree-based Model

Mohammad Febryan Khamim

1 Decision Tree

Decision Tree adalah algoritma *supervised learning* berbasis pohon (*tree-based*) yang digunakan untuk tugas **klasifikasi** dan **regresi**. ***Decision Tree*** atau pohon keputusan membantu untuk menentukan keputusan dengan menunjukkan perbedaan pilihan dan bagaimana keterkaitannya.

1.1 Elemen-Elemen dalam Decision Tree

Berikut adalah beberapa elemen-elemen dalam ***Decision Tree***, di antaranya adalah:

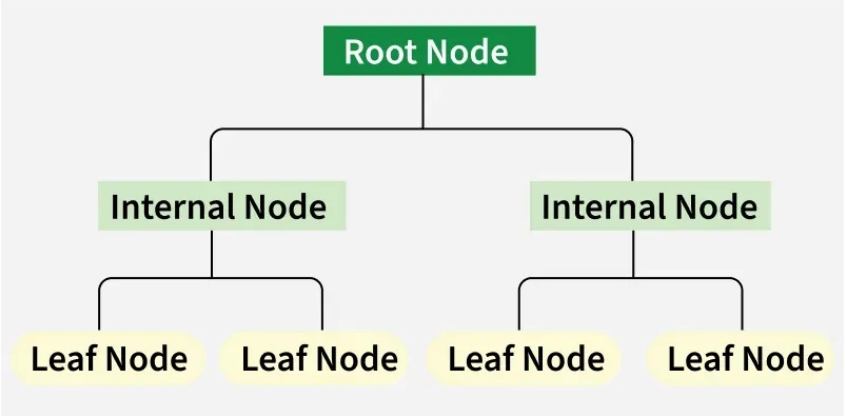

Figure 1: Ilustrasi Struktur Decision Tree

- **Root node** : Titik awal percabangan pohon atau pertanyaan utama.
- **Branches node** : Garis yang menghubungkan *nodes* dan menggambarkan alur dari keputusan satu ke keputusan selanjutnya.
- **Internal nodes** : Letak keputusan berdasarkan fitur.
- **Leaf nodes** : Titik paling akhir $\rightarrow$ letak keputusan.

1.2 Digunakan Ketika

- **Cocok digunakan** : Butuh model yang mudah dijelaskan, fitur campuran dan minim proses, *dataset* kecil-menengah.
- **Tidak digunakan** : *Dataset* besar, fitur kategori banyak.

1.3 Kelebihan dan Kekurangan Decision Tree

Kelebihan	Kekurangan
• Mudah dipahami • Dapat digunakan untuk klasifikasi atau regresi • Dapat menangkap berbagai jenis fitur • Menangkap hubungan nonlinier • Tahan terhadap *outlier* • Prediksi cepat • Mampu menangani *missing value*	• Mudah mengalami *overfitting* • Tidak stabil terhadap perubahan data • Bias terhadap fitur dengan banyak kategori • Bersifat *greedy* dan lokal optima • Kinerja regresi terbatas • Sensitif terhadap *imbalanced data*

1.4 Langkah Algoritma Decision Tree

Terdapat beberapa langkah dalam membentuk ***Decision Tree***, di antaranya sebagai berikut.

1. **Menentukan Gini Score Tiap Leaf**

$$\text{Gini} = 1 - \sum_{i=1}^{n} P_i^2$$

2. **Menentukan Total Gini Impurity**

$$\text{Total Gini} = \text{bobot}_1 \times \text{Gini}_1 + \text{bobot}_2 \times \text{Gini}_2$$

atau dapat dituliskan sebagai:

$$\text{Total Gini} = \text{Rata-rata Gini berbobot}$$

3. **Menggunakan fitur dengan Gini terkecil sebagai *root node***
4. **Ulangi langkah 1 – 3 hingga pohon terbentuk**

2 Random Forest

Random Forest adalah algoritma *tree-based* pada *supervised learning* yang menggunakan banyak ***Decision Tree*** untuk menuntukan keputusan. Setiap pohon memeriksa bagian acak yang berbeda dari data dan hasilnya digabungkan melalui pemungutan suara untuk menentukan keputusan yang sesuai.

Random Forest dikembangkan untuk mengatasi kelemahan utama dari *Decision Tree*, yaitu kecenderungan terhadap *overfitting* dan ketidakstabilan model. Alih-alih menggunakan satu pohon keputusan, *Random Forest* membangun banyak pohon yang saling independen dan mengombinasikan hasil prediksinya.

Dengan menggabungkan banyak pohon, kesalahan dari satu pohon dapat dikompensasi oleh pohon lainnya, sehingga model menjadi lebih stabil dan memiliki kemampuan generalisasi yang lebih baik terhadap data baru.

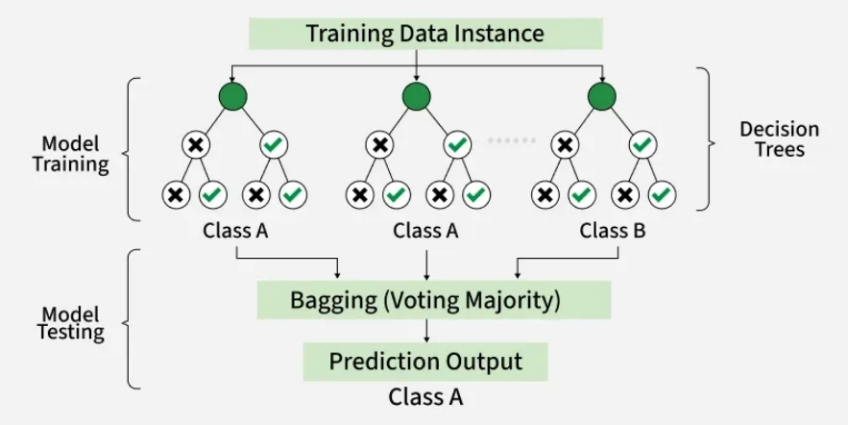

Figure 2: Ilustrasi Struktur ***Random Forest***

2.1 Prinsip Kerja *Random Forest*

Keacakan (*randomness*) pada *Random Forest* berasal dari dua aspek utama, yaitu pemilihan data secara acak melalui proses *bootstrap* dan pemilihan fitur secara acak pada setiap percabangan pohon. Kombinasi kedua mekanisme ini menghasilkan kumpulan pohon yang beragam dan tidak saling berkorelasi kuat.

Untuk tugas klasifikasi, hasil akhir ditentukan melalui mekanisme *majority voting*, sedangkan untuk regresi digunakan nilai rata-rata dari seluruh prediksi pohon.

- Membuat atau membentuk banyak ***Decision Tree***
- Memilih fitur-fitur secara acak
- Masing-masing *tree* membuat prediksi
- Mengombinasikan prediksi

2.2 Kelebihan dan Kekurangan *Random Forest*

Kelebihan	**Kekurangan**
• Akurat memprediksi *dataset* besar • Menangani *missing value* • Tidak memerlukan normalisasi dan standardisasi • Mengurangi *overfitting*	• Mahal dari segi komputasi • Sulit untuk diinterpretasikan

2.3 Parameter dalam *Random Forest*

Pengaturan parameter yang tepat sangat berpengaruh terhadap performa model *Random Forest*. Parameter seperti jumlah pohon, kedalaman pohon, dan jumlah fitur yang

dipilih perlu disesuaikan dengan karakteristik data agar diperoleh keseimbangan antara akurasi dan efisiensi komputasi. Terdapat beberapa parameter yang dapat diatur dalam ***Random Forest*** untuk meningkatkan performa dari model, di antaranya sebagai berikut.

1. **n_estimators**

   Jumlah pohon keputusan yang dibentuk dalam model. Semakin banyak jumlah pohon, umumnya akurasi model meningkat, namun kompleksitas komputasi juga bertambah. Sebaliknya, jumlah pohon yang terlalu sedikit dapat menurunkan akurasi model.

2. **max_features**

   Jumlah fitur yang dipilih secara acak saat mencari *split* terbaik pada setiap *node*. Parameter ini berfungsi untuk mengontrol *overfitting* dengan membatasi jumlah fitur yang dipertimbangkan. Beberapa pilihan nilai yang umum digunakan antara lain:

   - `"sqrt"` : $\sqrt{\text{jumlah fitur}}$ (rekomendasi umum)
   - `"log2"` : $\log_2$(jumlah fitur) untuk variasi yang lebih besar
   - `None` : menggunakan seluruh fitur

3. **max_depth**

   Kedalaman maksimum setiap pohon keputusan. Kedalaman yang terlalu kecil dapat menyebabkan *underfitting*, sedangkan kedalaman yang terlalu besar dapat menyebabkan *overfitting*.

4. **max_leaf_nodes**

   Batas maksimum jumlah daun (*leaf nodes*) yang diperbolehkan pada setiap pohon.

5. **max_samples**

   Jumlah atau proporsi sampel data yang diambil secara acak untuk membangun setiap pohon pada proses *bootstrap*.

6. **min_samples_split**

   Jumlah minimum sampel yang diperlukan untuk membagi (*split*) sebuah *node* menjadi dua cabang.

3 Source Code Python

Listing 1: Python Code

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Load the data
loans = pd.read_csv(r'e:\Perkuliahan\Karier\15-Decision-Trees-and-
    Random-Forests\loan_data.csv')
loans.info()
loans.describe()
loans.head()

# Exploratory Data Analysis
plt.figure(figsize=(10,6))
loans[loans['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue
    ',
                                              bins=30,label='Credit
                                                 .Policy=1')
loans[loans['credit.policy']==0]['fico'].hist(alpha=0.5,color='red'
    ,
                                              bins=30,label='Credit
                                                 .Policy=0')
plt.legend()
plt.xlabel('FICO')

plt.figure(figsize=(10,6))
loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5,color='
    blue',
                                              bins=30,label='Not
                                                 Fully Paid=1')
loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5,color='red
    ',
                                              bins=30,label='Not
                                                 Fully Paid=0')
plt.legend()
plt.xlabel('FICO')

countplot = sns.countplot(x='purpose',hue='not.fully.paid',data=
    loans,palette='Set1')
countplot.set_xticklabels(countplot.get_xticklabels(),rotation=45)
plt.xlabel('Purpose')
plt.ylabel('Count')
plt.figure(figsize=(11,7))
plt.show()

jointplot = sns.jointplot(x='fico',y='int.rate',data=loans,color='
```

```
   purple')

plt.figure(figsize=(11,7))
sns.lmplot(y='int.rate',x='fico',data=loans,hue='credit.policy',
           col='not.fully.paid',palette='Set1')

# Data Preprocessing
cat_feats = ['purpose']
final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True
   )

# Train Test Split
from sklearn.model_selection import train_test_split
X = final_data.drop('not.fully.paid',axis=1)
y = final_data['not.fully.paid']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
   =0.3, random_state=42)

# Training a Decision Tree Model
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
predictions = dtree.predict(X_test)

# Evaluation
from sklearn.metrics import classification_report,confusion_matrix

print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))

# Training a Random Forest Model
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=600)
rfc.fit(X_train,y_train)
predictions = rfc.predict(X_test)

# Evaluation
from sklearn.metrics import classification_report,confusion_matrix

print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```