

Algoritma Random Forest

Oleh Kelompok 5

Anggota Kelompok

Alfonsus William

Hamonangan Sinaga

234311005

Mohammad Fakhriza

Maftukhin

234311018

Muhammad Aulia

Alfarouq

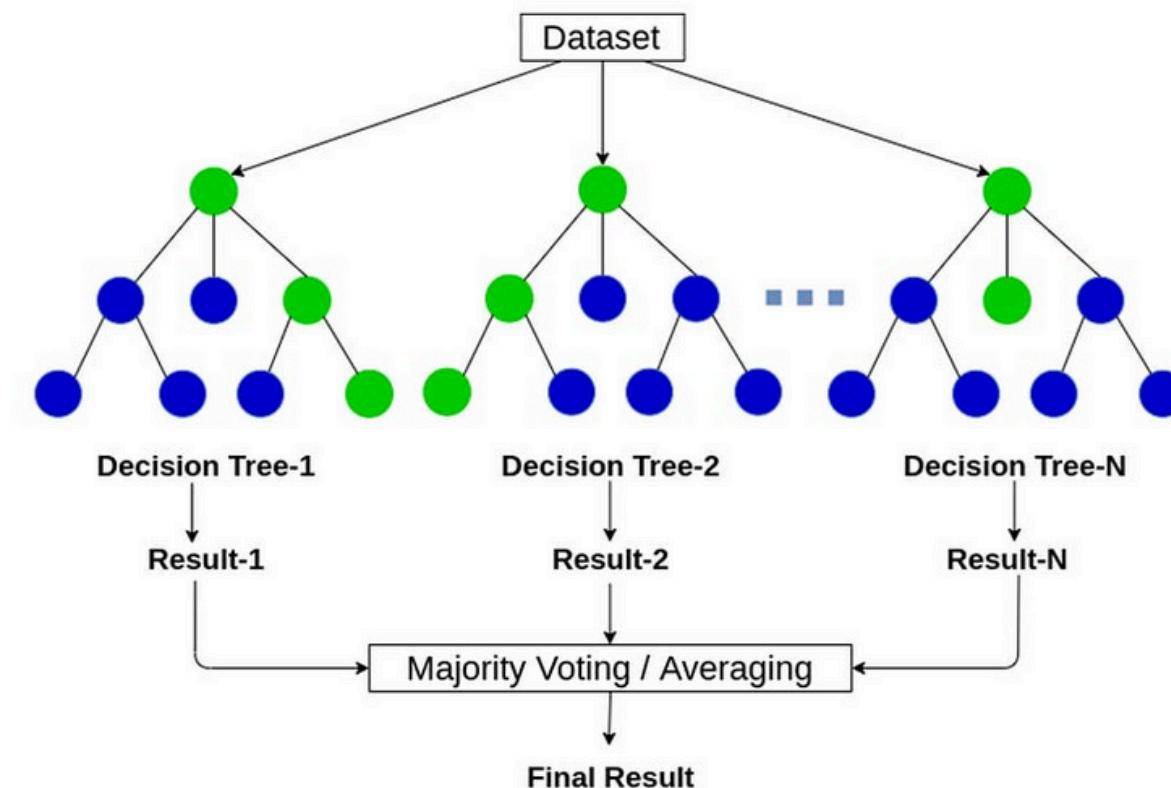
234311020

Definisi Random Forest

Random Forest adalah algoritma **supervised learning** yang termasuk dalam metode **ensemble learning**. Ensemble learning adalah menggabungkan beberapa model pengklasifikasi yang lebih lemah untuk memecahkan masalah yang lebih kompleks

- » Bekerja dengan membangun sejumlah besar *decision trees* (pohon keputusan) pada saat training.
- » Untuk **klasifikasi**, hasil akhirnya adalah *voting* (suara terbanyak) dari semua pohon.
- » Untuk **regresi**, hasil akhirnya adalah *rata-rata* (average) dari semua pohon.
- » Menggabungkan banyak model (pohon) untuk menghasilkan prediksi yang lebih akurat dan stabil.

Random Forest



Konsep Inti



Ensemble Learning

Ide menggabungkan beberapa model (disebut "weak learners") untuk menciptakan satu model ("strong learner") yang lebih kuat dan akurat.



Bagging

(Bootstrap Aggregating). Membuat banyak subset data secara acak dari data training (dengan *replacement*) untuk melatih setiap pohon secara independen.



Random Subspace

Saat membelah *node*, algoritma hanya mempertimbangkan subset fitur acak, bukan semua fitur. Ini memastikan pohon-pohnnya beragam (tidak identik).

Hipotesis Function($H(x)$)

"Hipotesis function $h(x)$ adalah prediksi yang dibuat oleh satu pohon dalam Random Forest. Ia bekerja berdasarkan aturan IF-THEN yang ada di dalam pohon keputusan. Jadi setiap pohon punya pendapat sendiri dulu, dan pendapat tiap pohon inilah yang nantinya digabungkan."

Pohon Tunggal ($h(x)$)

Fungsi hipotesis untuk satu *decision tree* adalah serangkaian aturan IF-THEN yang mempartisi ruang fitur.

Forest ($H(x)$)

Hipotesis untuk *Random Forest* adalah agregasi dari semua pohon (K). Ini adalah fungsi non-linear yang kompleks.

Klasifikasi (Voting):

$$H(x) = \text{mode} \{ h_1(x), h_2(x), \dots, h_K(x) \}$$

$H(x)$

→ Prediksi akhir dari keseluruhan Random Forest untuk input x . Ini adalah hasil gabungan dari semua pohon.

$h_1(x), h_2(x), \dots, h_k(x)$

→ Prediksi dari masing-masing pohon k dalam hutan (forest). Setiap pohon memberikan satu suara/class prediction untuk data x .

mode{...}

→ Nilai yang paling sering muncul (majoritas). Dalam klasifikasi, Random Forest mengambil hasil voting terbanyak dari semua pohon.

K

→ Jumlah total pohon dalam forest.

Cost Function (Kriteria Split)

Random Forest tidak memiliki satu "cost function" global. Sebaliknya, algoritma ini menggunakan **impurity measures** (pengukur ketidakmurnian) di setiap *node* untuk memutuskan *split* (pembelahan) terbaik.

Klasifikasi: Gini Impurity

Mengukur seberapa sering elemen yang dipilih secara acak akan salah diklasifikasikan. Tujuannya adalah meminimalkan Gini.

$$G = 1 - \sum_{i=1}^C (p_i)^2$$

- C
Jumlah kelas dalam node (misalnya: Churn & Not Churn \rightarrow C = 2).

- p_i
Proporsi data untuk kelas ke-i pada node tersebut.
Contoh: jika 7 dari 10 data adalah "Not Churn", maka $p = 0.7$.

- p_i^2
Probabilitas tersebut dikuadratkan untuk mengukur "kemurnian" kelas.

- Σ (sigma)
Menjumlahkan seluruh p_i^2 dari semua kelas.

- $1 - \Sigma p_i^2$
Mengubah hasilnya menjadi ukuran impurity.

\rightarrow Semakin kecil nilai Gini, semakin murni node (lebih sedikit campuran kelas).

Regresi: Mean Squared Error (MSE)

Digunakan untuk regresi. Memilih split yang paling mengurangi varians (atau MSE) dalam node anak.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y})^2$$

- n
Jumlah total data dalam node.
- y_i
Nilai asli (ground truth) pada data ke-i.
- \hat{y} (y-hat)
Nilai prediksi di node tersebut (biasanya rata-rata dari nilai-nilai dalam node).
- $(y_i - \hat{y})$
Selisih antara nilai asli dan prediksi.

- $(y_i - \hat{y})^2$
- Selisihnya dikuadratkan agar tidak bernilai negatif
- dan memberi penalti lebih besar untuk error besar.
- Σ (sigma)
- Menjumlahkan seluruh error kuadrat.
- $1/n$
- Mengambil rata-rata dari semua error.

Hyperparameter Penting



n_estimators

Jumlah pohon (trees) dalam forest. Semakin banyak, performa semakin baik dan stabil, namun *training* lebih lama.



max_depth

Kedalaman maksimum setiap pohon. Jika terlalu dalam, bisa *overfitting*. Jika terlalu dangkal, bisa *underfitting*.



min_samples_split

Jumlah minimum sampel yang diperlukan untuk membelah (split) sebuah *internal node*. Berguna untuk mengontrol *overfitting*.



max_features

Jumlah fitur yang dipertimbangkan secara acak saat mencari *split* terbaik. Kunci untuk mengurangi korelasi antar pohon.

Kelebihan dan Kekurangan Random Forest

👍 Kelebihan

Akurasi Tinggi: Menggunakan voting dari banyak pohon sehingga hasil lebih stabil dan seringkali lebih unggul dari model lain.

Mencegah Overfitting: Teknik Bagging dan Random Subspace memastikan variasi antar pohon, mencegah model terlalu spesifik pada data latih.

Robust terhadap Noise: Kurang sensitif terhadap data outlier (pencilan) dan data yang noisy (berisik).

Feature Importance: Secara otomatis menyediakan metrik seberapa penting setiap fitur dalam prediksi.

👎 Kekurangan

Komputasi Mahal: Membangun ratusan pohon membutuhkan waktu training yang lebih lama dan sumber daya komputasi yang besar.

Kurang Interpretatif (Black Box): Karena agregasi dari banyak pohon, sulit untuk menjelaskan *mengapa* model membuat prediksi tertentu (dibanding Decision Tree tunggal).

Lambat untuk Real-Time: Waktu prediksi (inference) bisa lebih lambat karena harus melewati semua pohon untuk mendapatkan hasil voting.

Kapan Menggunakan & Menghindari Random Forest

★ Waktu Terbaik Penggunaan

Membutuhkan Akurasi Maksimal: Ketika akurasi prediksi adalah prioritas utama dan Anda memiliki sumber daya komputasi yang memadai.

Data Beragam/Noisy: Ketika dataset Anda mengandung banyak fitur (dimensi tinggi), data kategorikal, atau memiliki banyak *noise* atau *outlier*.

Tidak Perlu Normalisasi: Ketika Anda ingin menghindari proses scaling/normalisasi data karena RF tidak sensitif terhadap skala fitur.

Memerlukan Feature Ranking: Ketika Anda perlu mengetahui fitur mana yang paling penting dalam memprediksi target.

⚠ Kapan Harus Menghindari

Interpretasi Mutlak Penting: Ketika model harus bisa dijelaskan secara sederhana ke stakeholder non-teknis (misalnya, di industri finansial atau medis).

Keterbatasan Sumber Daya: Ketika Anda bekerja pada sistem dengan RAM atau CPU yang sangat terbatas, terutama untuk data yang sangat besar.

Kebutuhan Prediksi Real-Time: Untuk aplikasi yang membutuhkan kecepatan prediksi sangat tinggi (latency rendah), karena RF harus menjalankan ratusan prediksi per *instance*.

Data Sangat Langka (Sparse): Untuk data teks atau data yang sebagian besar berisi nol (0), model linier atau algoritma lain mungkin lebih efisien.

Contoh Kasus: Prediksi Churn Pelanggan

Studi Kasus: Latar Belakang

Skenario: Data `customer_churn_light.csv`

Memprediksi pelanggan mana yang akan *churn* (berhenti) berdasarkan data penggunaan mereka.

» Target: `churn` (Yes/No)

» Fitur Numerik:

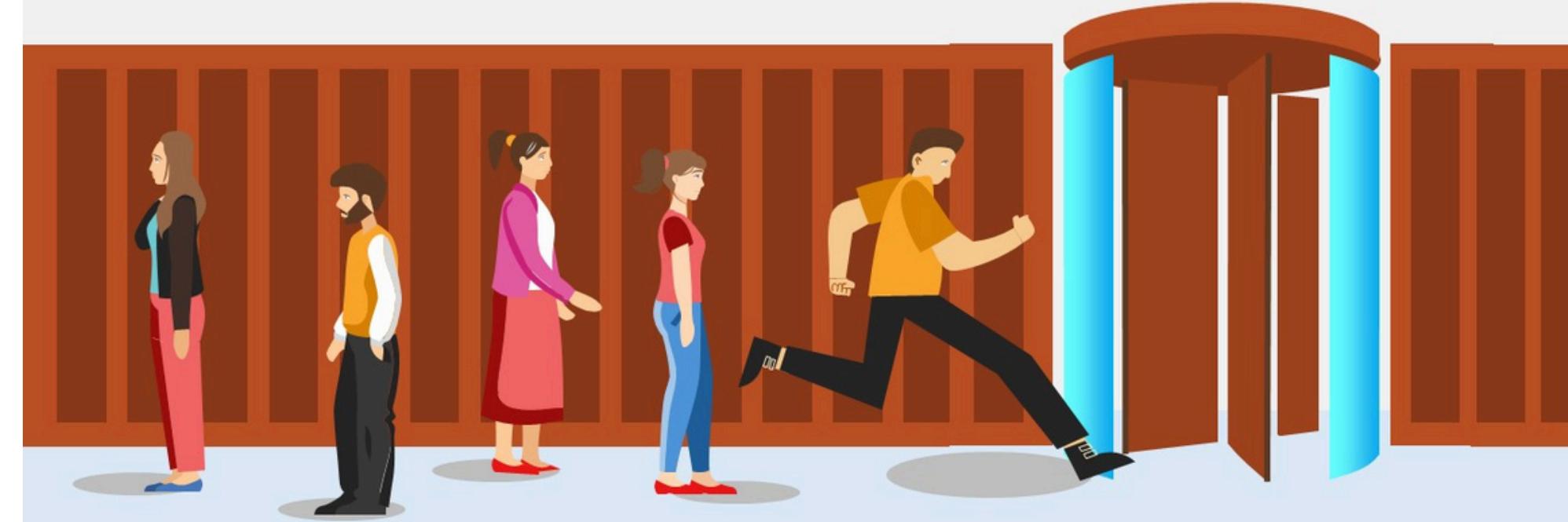
- `tenure_months` (lama berlangganan)
- `monthly_usage` (penggunaan bulanan)
- `complaints` (jumlah komplain)
- `payment_late` (keterlambatan bayar)

» Fitur Kategorikal:

- `plan_type` (jenis paket)

CUSTOMER CHURN

Customer churn refers to the rate at which customers stop doing business with a company. High churn can impact revenue and growth. Businesses must identify reasons for churn, such as poor service or better competitors. Reducing churn involves improving customer experience and offering personalized solutions. Understanding customer needs helps build long-term loyalty and retention.



Bagaimana Random Forest Bekerja



Preprocessing Pipeline: Data `plan_type` (teks, mis: "Basic", "Premium") diubah menjadi kolom numerik (`plan_type_Basic`, `plan_type_Premium`) oleh OneHotEncoder secara otomatis.



Bangun 200 Pohon: Model membangun 200 pohon. Setiap pohon dilatih pada sampel acak (subset) dari data pelanggan (`X_train`).



Contoh Split Pohon: Sebuah pohon mungkin fokus pada fitur numerik, misal: `tenure_months < 6` DAN `complaints > 1` untuk memprediksi "Churn".



Voting Akhir: Model melakukan voting pada data `X_test`. Jika 180 dari 200 pohon memprediksi "Churn", maka prediksi akhir adalah "Churn".

Penjelasan Kode Program (Python)

Kode: 1. Persiapan & Pembagian Data

Kode: Memuat & Membagi Data

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Muat data dari file csv
df = pd.read_csv('customer_churn_light.csv')

# 1. Pisahkan Fitur (X) dan Target (y)
X = df.drop('churn', axis=1)
y = df['churn']

# 2. Bagi data menjadi set Latih dan Tes
# (80% latih, 20% tes)
# stratify=y memastikan proporsi
# 'churn' sama di data latih & tes
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2,
    random_state=42, stratify=y
)
```

Penjelasan

- » `X` berisi semua kolom *kecuali* `churn` (fitur/pertanyaan).
- » `y` hanya berisi kolom `churn` (target/jawaban).
- » `train_test_split` adalah langkah krusial. Kita "menyembunyikan" 20% data (`X_test`, `y_test`) dari model. Model hanya boleh belajar (`fit`) pada `X_train` dan `y_train`.

Kode: 2. Definisi Preprocessing & Model

Kode: Mendefinisikan Komponen

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier

# Tentukan kolom-kolom
categorical_cols = ['plan_type']
# (numeric_cols tidak perlu didefinisikan
# jika kita pakai remainder='passthrough')

# 3. Buat Preprocessor
# OneHotEncoder mengubah data teks (plan_type)
# menjadi angka (0 atau 1)
preprocess = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'),
     categorical_cols),
], remainder='passthrough')

# 4. Tentukan Model
model = RandomForestClassifier(n_estimators=200,
                               random_state=42)
```

Penjelasan

- » `ColumnTransformer` adalah "otak" preprocessing. kelas ini tahu bahwa harus menerapkan `OneHotEncoder` ke `categorical_cols` dan membiarkan (`passthrough`) sisa kolom lainnya.
- » `RandomForestClassifier` didefinisikan untuk menggunakan 200 pohon.

Kode: 3. Training & Evaluasi Pipeline

Kode: Menjalankan Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

# 5. Buat Pipeline Utama
# Menggabungkan langkah (3) dan (4)
pipeline = Pipeline([
    ('prep', preprocess),
    ('rf', model)
])

# 6. Latih (Fit) Pipeline
# Pipeline secara otomatis menerapkan
# 'prep' SEBELUM melatih 'rf'
pipeline.fit(X_train, y_train)

# 7. Prediksi & Evaluasi
preds = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, preds)

print("Accuracy:", accuracy)
```

Penjelasan

- » `Pipeline` menggabungkan `preprocess` dan `model` menjadi satu objek.
- » `pipeline.fit(X_train, ...)`: Ini adalah perintah utamanya. Pipeline secara otomatis menerapkan preprocessing ke `X_train`, lalu melatih model RF pada data yang sudah bersih itu.
- » `pipeline.predict(X_test)`: Pipeline menerapkan preprocessing yang *sama* ke `X_test` lalu membuat prediksi.
- » `accuracy_score` memvalidasi hasil kita.

Hasil Output Program

Console Output

```
Accuracy: 1.0
      customer_id  tenure_months  monthly_usage  complaints  payment_late \
5              6                  15            12.5          0             1
1              2                  3             2.1          1             1

plan_type
5   Premium
1   Basic
['No' 'Yes']
5   No
1   Yes
Name: churn, dtype: object
```

Interpretasi Hasil

- » **Akurasi 1.0:** Model memprediksi data uji dengan akurasi 100% sempurna pada sampel ini.
- » **Data Sampel:**
 - Pelanggan ID 6 (Tenure 15 bulan, Usage 12.5)
 - Pelanggan ID 2 (Tenure 3 bulan, Usage 2.1)
- » **Hasil Prediksi:**
 - **ID 6 (No):** Diprediksi setia (tidak churn).
 - **ID 2 (Yes):** Diprediksi akan berhenti (churn). Kemungkinan karena masa langganan pendek dan penggunaan rendah.

Terima Kasih