Random Forest

https://github.com/as-budi/Embedded_AI.git

1. Pengertian Random Forest

- Random Forest adalah algoritma machine learning berbasis ensemble yang digunakan untuk tugas klasifikasi dan regresi.
- Algoritma ini terdiri dari kumpulan pohon keputusan (*decision trees*) yang bekerja bersama untuk meningkatkan akurasi dan mengurangi overfitting dibandingkan dengan pohon keputusan tunggal.
- Metode ini diperkenalkan oleh **Leo Breiman** pada tahun 2001 dan didasarkan pada konsep **Bagging (Bootstrap Aggregating)**, yang bertujuan untuk meningkatkan performa model dengan menggabungkan hasil prediksi dari beberapa pohon keputusan.

2. Cara Kerja Random Forest

- Random Forest bekerja dengan membangun banyak pohon keputusan dari subset data yang berbeda dan kemudian menggabungkan hasilnya untuk mendapatkan prediksi akhir.

3. Tahapan dalam Random Forest:

1. **Bootstrap Sampling**
    - Dari dataset awal, **N** sampel dipilih secara acak dengan pengembalian (bootstrap).
    - Artinya, beberapa sampel bisa terpilih lebih dari sekali, sedangkan beberapa lainnya mungkin tidak terpilih.

2. **Pembentukan Pohon Keputusan**

- Setiap pohon dibuat menggunakan subset data hasil bootstrap.
- Pada setiap percabangan dalam pohon, hanya sejumlah fitur yang dipilih secara acak untuk dipertimbangkan sebagai kandidat pemisahan (**feature randomness**).
- Pohon tumbuh hingga kedalaman tertentu tanpa pemangkasan (*pruning*).

3. **Penggabungan Hasil**
   - Untuk klasifikasi: hasil prediksi setiap pohon dihitung berdasarkan **voting mayoritas**.
   - Untuk regresi: rata-rata dari prediksi setiap pohon digunakan sebagai hasil akhir.

4. Kelebihan dan Kekurangan Random Forest

Kelebihan:

✓ Akurasi Tinggi

- Mengurangi overfitting dibandingkan dengan pohon keputusan tunggal.
- Lebih tahan terhadap perubahan kecil dalam dataset.

✓ Mampu Menangani Data dengan Banyak Fitur

- Secara otomatis melakukan seleksi fitur dengan memilih subset fitur secara acak pada setiap pohon.

✓ Dapat Digunakan untuk Klasifikasi dan Regresi

- Fleksibel untuk berbagai jenis tugas machine learning.

✓ Tidak Sensitif terhadap Data Hilang

- Dapat menangani missing values dan tetap bekerja dengan baik.

✓ Menangani Data Non-Linear

- Berkat kombinasi banyak pohon, model ini dapat menangkap hubungan non-linear dalam data.

Kekurangan:

❌ Kompleksitas Tinggi

- Membutuhkan lebih banyak waktu dan sumber daya dibandingkan decision tree tunggal.

❌ Kurang Interpretable

- Karena terdiri dari banyak pohon, sulit untuk memahami bagaimana model mengambil keputusan.

✕ Menggunakan Banyak Memori

- Penyimpanan model besar, terutama jika jumlah pohon tinggi.

5. Contoh Implementasi dalam Python

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load dataset Iris
iris = load_iris()
X = iris.data
y = iris.target

# Bagi dataset menjadi training dan testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Buat model Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Prediksi pada data uji
y_pred = rf.predict(X_test)

# Evaluasi model
accuracy = accuracy_score(y_test, y_pred)
print(f'Akurasi Model: {accuracy:.2f}')
```

6. Parameter Penting dalam Random Forest

Beberapa parameter penting dalam Random Forest yang dapat dikonfigurasi:

- `n_estimators` → Jumlah pohon dalam hutan (semakin banyak, semakin stabil).
- `max_features` → Jumlah fitur yang dipilih secara acak di setiap percabangan.
- `max_depth` → Kedalaman maksimum pohon keputusan.
- `min_samples_split` → Jumlah sampel minimal untuk membagi simpul (node).
- `min_samples_leaf` → Jumlah minimum sampel di setiap daun (*leaf*

7. Contoh Perhitungan Manual Random Forest

- Misalkan kita memiliki dataset **Iris**, tetapi untuk penyederhanaan, kita hanya akan menggunakan **4 sampel dan 2 fitur**:

Sample	Panjang Kelopak	Lebar Kelopak	Class (Target)
A	5.1	3.5	0 (Setosa)
B	4.9	3.0	0 (Setosa)
C	6.2	3.4	1 (Versicolor)
D	5.8	2.7	1 (Versicolor)

- Target (kelas) yang akan diprediksi: **0 (Setosa), 1 (Versicolor)**

Bootstrap Sampling

- Dalam **Random Forest**, kita akan mengambil beberapa sampel secara acak dengan pengembalian (bootstrap sampling).
- Misalkan kita membangun **3 pohon keputusan**, dan masing-masing mengambil subset berbeda.

Tree 1 (Subset 1)

Sample	Feature 1	Feature 2	Class
A	5.1	3.5	0
B	4.9	3.0	0
C	6.2	3.4	1

Tree 2 (Subset 2)

Sample	Feature 1	Feature 2	Class
A	5.1	3.5	0
D	5.8	2.7	1
C	6.2	3.4	1

Tree 3 (Subset 3)

Sample	Feature 1	Feature 2	Class
B	4.9	3.0	0
D	5.8	2.7	1
C	6.2	3.4	1

Pembentukan Pohon Keputusan

- Setiap pohon akan membuat aturan berdasarkan **pembagian optimal (splitting criterion)**.
- Dalam Random Forest, pembagian didasarkan pada **Gini Impurity** atau **Entropy**.

Gini Impurity Formula

- $Gini = 1 - \sum p_i^2$
- di mana p_i adalah probabilitas masing-masing kelas dalam node.

Contoh Perhitungan Gini Impurity

- Misalkan kita memiliki node awal dengan dua kelas untuk **Tree 1**, masing-masing dengan probabilitas:
  - **Setosa**: 2/3 = 0.67
  - **Versicolor**: 1/3 = 0.33
- Maka Gini impurity sebelum pembagian:
- $Gini_{awal} = 1 - (0.67^2 + 0.33^2) = 1 - (0.4489 + 0.1089) = 0.442$

- Setelah pemisahan pada **Feature 1 (Panjang Kelopak)** dengan batas **5.5**, kita memperoleh dua kelompok:
  1. **Kiri (Feature 1 ≤ 5.5)** → {A (Setosa), B (Setosa)}
  2. **Kanan (Feature 1 > 5.5)** → {C (Versicolor)}
- Gini impurity masing-masing node:
  - **Gini kiri**: Semua kelas Setosa → $Gini = 1 - (1^2) = 0$
  - **Gini kanan**: Semua kelas Versicolor → $Gini = 1 - (1^2) = 0$

- Gini impurity total dihitung dengan formula:
- $Gini_{total} = \frac{n_{kiri}}{n_{total}} Gini_{kiri} + \frac{n_{kanan}}{n_{total}} Gini_{kanan}$
- $Gini_{total} = \frac{2}{3}(0) + \frac{1}{3}(0) = 0$
- Karena Gini impurity setelah pemisahan **lebih kecil**, maka kita memilih **Feature 1 dengan threshold 5.5** sebagai split terbaik.
- Perlakkuan yang sama digunakan untuk **Tree 2** dan **Tree 3**.

Prediksi dengan Voting

Misalkan kita memiliki sampel baru dengan **Feature 1 = 5.0, Feature 2 = 3.2**, dan kita ingin memprediksi kelasnya.

- **Tree 1**: Memilih **Setosa (0)**
- **Tree 2**: Memilih **Setosa (0)**
- **Tree 3**: Memilih **Setosa (0)**

Karena mayoritas pohon memilih **Setosa (0)**, maka hasil akhir model adalah **Setosa (0)**.

8. Generalisasi ke Dataset Lebih Besar

- Dalam implementasi nyata seperti pada dataset **Iris** dengan **150 sampel dan 4 fitur**, proses ini berulang dalam skala yang lebih besar:
- Setiap **tree** menggunakan subset data (bootstrap sample).
- Fitur dipilih secara acak untuk **pembagian optimal**.
- **Hasil voting** dari semua pohon menentukan prediksi akhir.
- Jika kita mengulangi proses ini untuk 100 pohon, maka hasil akhir akan menjadi lebih stabil dan akurat dibandingkan hanya menggunakan satu pohon keputusan.

Kesimpulan

1. **Random Forest membangun banyak pohon keputusan dengan bootstrap sampling.**
2. **Setiap pohon memilih subset fitur acak untuk menghindari overfitting.**
3. **Pemisahan dalam pohon menggunakan kriteria seperti Gini Impurity atau Entropy.**
4. **Prediksi akhir diperoleh melalui voting mayoritas (klasifikasi) atau rata-rata (regresi).**
5. **Dengan lebih banyak pohon, model menjadi lebih stabil dan lebih akurat.**