1. Cara kerja algoritma Classification and Regression Tree dari scratch pada repository ini

a. Inisialisasi model dan parameter

Proses dimulai dengan menginisialisasi parameter model berupa min_samples, yaitu jumlah minimum sampel yang diperlukan untuk melakukan *split* di *node*, dan max_height, yang menentukan kedalaman maksimum pohon. Model diinisialisasi dengan *tree* sebagai *None*, yang nantinya akan diisi dengan struktur pohon hasil pelatihan.

b. Proses *fitting* data

Ketika fit(X, y) dipanggil, algoritma mulai membangun pohon keputusan dengan memanggil *method* grow_tree(X, y). Di sini, data input X dan label y digunakan untuk membangun pohon secara rekursif. Jika *node* memenuhi syarat untuk menjadi *node* internal (jumlah sampel cukup dan kedalaman belum mencapai maksimum), algoritma akan mencari split terbaik untuk membagi data.

c. Menumbuhkan pohon (*Grow Tree*)

Dalam method grow_tree, algoritma menentukan apakah *node* saat ini akan menjadi *node* internal atau *leaf*. Jika jumlah sampel memenuhi syarat dan kedalaman pohon belum mencapai batas, algoritma mencari *split* terbaik dengan memanggil find_split. Jika *split* yang valid ditemukan dengan informasi gain yang positif, data dipecah menjadi dua subset untuk *node* kiri dan kanan. Kemudian proses rekursif dilanjutkan untuk setiap subset tersebut.

d. Menemukan *split* terbaik

Method find_split mengevaluasi semua fitur dan *threshold* yang mungkin untuk menemukan *split* yang memberikan Gain informasi tertinggi. Gain dihitung menggunakan penurunan Gini Impurity. Gini Impurity mengukur ketidakmurnian *node* dan dihitung dengan formula:

$$\text{Gini}(y) = 1 - \sum_k p_k^2$$

di mana p_k adalah proporsi sampel dengan kelas *k* di dalam *node*. Setelah Gini dihitung sebelum dan sesudah split, Gain dihitung sebagai:

$$\text{Gain} = \text{Gini}(y) - \left(\frac{n_{\text{left}}}{n} \times \text{Gini}(y_{\text{left}}) + \frac{n_{\text{right}}}{n} \times \text{Gini}(y_{\text{right}}) \right)$$

di mana n_{left} serta n_{right} adalah jumlah sampel di *node* kiri dan kanan setelah di *split*.

e. Pembagian data dan perhitungan gain

Jika *split* yang valid ditemukan, *method* partition digunakan untuk memisahkan data menjadi dua subset berdasarkan *threshold* yang dipilih. Gain dihitung sebagai perbedaan antara impurity node sebelum dan sesudah split dengan tujuan untuk memilih *split* yang memaksimalkan pemisahan antara kelas.

f. Membentuk *leaf node*

Jika tidak ada *split* yang memberikan gain positif atau jika pohon telah mencapai

kedalaman maksimum, *node* tersebut menjadi *leaf*. Nilai *leaf* ditentukan oleh kelas mayoritas dari sampel di *node* tersebut, yang dihitung menggunakan determine_leaf_value.

g. Proses prediksi
   Setelah pohon selesai dibangun, model dapat digunakan untuk memprediksi kelas data baru dengan *method* predict. Untuk setiap sampel data baru, algoritma akan mengikuti jalur dari *root* ke *leaf* berdasarkan nilai fitur dan *threshold* di setiap *node*. Prediksi akhir ditentukan oleh nilai *leaf node* yang tercapai.
h. Prediksi berdasarkan *node*
   Saat melakukan prediksi, *method* make_prediction dimulai dari *root node* dan bergerak melalui pohon sesuai dengan *threshold* yang ditetapkan di setiap *node* sampai mencapai *leaf node*. Nilai dari *leaf node* ini adalah hasil prediksi untuk sampel tersebut.

4. Perbandingan hasil evaluasi model

Hasil evaluasi model menunjukkan bahwa performa **CARTScratch** yang diimplementasikan dari scratch dan **DecisionTreeClassifier** dari scikit-learn hampir identik. Kedua model memiliki Akurasi dan F1 Score yang sama pada set test, yaitu masing-masing 0.80 dan 0.62, serta hasil Cross-Validation yang serupa dengan Cross-Validation Accuracy 0.79 ± 0.01 dan F1 Score 0.58 ± 0.03.

Kesamaan ini terjadi karena penggunaan algoritma dasar CART yang sama di kedua model termasuk parameter seperti kedalaman maksimum pohon yang juga serupa. Algoritma CART sendiri cukup stabil, sehingga dengan parameter dan data yang sama, baik implementasi dari scratch maupun versi scikit-learn memberikan hasil yang konsisten dan hampir tidak ada perbedaan dalam konteks performa.

5. Improvement yang bisa dilakukan

Untuk meningkatkan kinerja model CART, ada beberapa ide perbaikan yang mungkin bisa dilakukan. Salah satu cara yang efektif adalah dengan menerapkan **pruning** pada pohon keputusan. Pruning membantu mengurangi kompleksitas pohon dengan memangkas cabang-cabang yang kurang penting, sehingga model menjadi lebih sederhana dan lebih baik dalam generalisasi data, mengurangi risiko *overfitting*. Selain itu, melakukan **hyperparameter tuning** juga bisa memberikan peningkatan signifikan. Menyesuaikan parameter seperti kedalaman maksimum pohon (max_height) atau jumlah sampel minimum untuk split (min_samples) melalui *grid search* atau *random search* dapat menemukan setelan optimal yang menghasilkan performa terbaik. Penggunaan **ensemble methods** seperti *Random Forest* juga dapat menjadi opsi yang baik, di mana beberapa pohon keputusan digabungkan untuk memperbaiki stabilitas dan akurasi prediksi. Dengan menggabungkan beberapa model CART, *ensemble methods* mungkin dapat menangkap lebih banyak pola dalam data, meningkatkan

generalisasi, dan mengurangi variabilitas hasil sehingga menghasilkan model yang lebih kuat dan akurat.