1. Cara kerja algoritma KNN dari *scratch* pada *repository* ini

a. Pertama, kita perlu menentukan nilai K, yaitu jumlah tetangga terdekat yang akan digunakan untuk menentukan kelas atau nilai prediksi. Selain itu, kita juga perlu memilih jenis metrik jarak yang akan digunakan untuk mengukur jarak antara dua titik. Metrik jarak yang diimplementasikan menyesuaikan dengan spesifikasi, yaitu Euclidean, Manhattan, dan Minkowski.
b. Setelah itu, semua data pelatihan berupa fitur ('X_train') dan labelnya ('y_train') disimpan dalam memori. Proses tersebut terjadi ketika *method* fit dipanggil. Pada tahap ini tidak ada proses pelatihan yang sebenarnya dilakukan. Implementasi *method* fit digunakan hanya untuk menyimpan data pelatihan ini untuk digunakan nanti saat melakukan prediksi.
c. Untuk setiap titik data uji, algoritma KNN akan melakukan serangkaian langkah berikut :
    - Menghitung jarak antara titik data uji dengan setiap titik data pelatihan. Misalnya, jika menggunakan Euclidean, jarak antara dua titik dihitung dengan rumus :

$$Jarak = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

    - Setelah menghitung semua jarak, algoritma mengurutkan data pelatihan berdasarkan jarak terdekat dari titik data uji. Lalu, **K** data terdekat (tetangga terdekat) dipilih.
    - Dari **K** tetangga terdekat ini, label-labelnya diambil. Misal, jika K=3 dan label-label tetangga terdekat adalah [0, 1, 0], maka label yang paling umum di antara tetangga adalah 0.
    - Untuk tugas klasifikasi, algoritma menentukan kelas yang paling sering muncul di antara **K** tetangga terdekat sebagai prediksi. Jika lebih banyak tetangga terdekat memiliki kelas tertentu, maka kelas itu yang akan dipilih. Sementara untuk tugas regresi, rata-rata nilai dari **K** tetangga terdekat diambil sebagai hasil prediksi.

d. Proses di atas diulang untuk setiap titik data dalam set data uji sampai menghasilkan serangkaian prediksi akhir untuk masing-masing titik.
e. Setelah seluruh data uji diproses, hasil akhirnya akan berupa daftar prediksi yang pada kasus *repository* ini adalah label kelas.

4. Perbandingan hasil evaluasi model

Berdasarkan hasil evaluasi model, performa **KNNScratch** yang diimplementasikan dari scratch dan **KNN** dari scikit-learn menunjukkan hasil yang hampir identik. Kedua model memiliki **Cross-Validation Accuracy** sebesar **0.77 ± 0.01** dan **Cross-Validation F1 Score** sebesar **0.55 ± 0.01**.

Hasil ini menunjukkan bahwa implementasi KNN dari scratch sudah cukup baik dan sejalan dengan versi scikit-learn. Karena KNN adalah algoritma yang relatif sederhana dan hanya melibatkan perhitungan jarak antara titik data untuk memilih tetangga terdekat, hasil implementasi dari scratch tidak terlalu berbeda dengan hasil dari library seperti scikit-learn.

Selama perhitungan jarak, nilai **K**, dan data yang digunakan tetap konsisten, hasil prediksi yang dihasilkan oleh kedua implementasi harusnya akan mirip.

Akan tetapi apabila ada perbedaan kecil dalam hasil jika dicoba pada dataset lain, itu mungkin disebabkan oleh cara library scikit-learn mengoptimalkan proses atau menangani situasi khusus, seperti jika ada beberapa tetangga dengan jarak yang sama.

5. *Improvement yang bisa dilakukan*

Mungkin bisa dicoba menggunakan struktur data seperti **KD-Tree** atau **Ball Tree** untuk mempercepat pencarian tetangga terdekat, terutama jika datanya besar atau memiliki banyak fitur. Memilih metrik jarak yang sesuai seperti **Cosine Similarity** atau **Mahalanobis Distance** sepertinya juga bisa membantu meningkatkan hasil, tergantung pada jenis data juga. Selain itu, bisa juga dengan melakukan penyetelan hyperparameter, seperti menentukan nilai **K** yang tepat melalui Cross-Validation Grid Search, bisa menjadi langkah yang baik untuk menemukan konfigurasi terbaik. Jadi, ada 3 opsi cara yang bisa dilakukan untuk melakukan *improvement* yaitu dengan mengoptimalkan struktur data, memilih metrik jarak yang tepat, dan melakukan penyetelan hyperparameter.