1. Cara kerja algoritma Logistic Regression dari scratch pada repository ini

a. Inisialisasi parameter

Proses dimulai dengan menginisialisasi parameter model, yaitu bobot (**w**) yang diatur sebagai array nol dengan panjang yang sama dengan jumlah fitur dan bias (**b**) yang juga diinisialisasi dengan nilai nol. Parameter lain, seperti *learning rate*, jumlah iterasi, jenis regularisasi, dan jenis *loss function* juga ditentukan di awal atau memiliki nilai *default*. Akan tetapi, tentu saja nilai tersebut bisa dimodifikasi tergantung dengan kebutuhan.

b. Perhitungan skor linear dan aplikasi fungsi sigmoid

Pada setiap iterasi, model menghitung skor linear dari fitur input menggunakan bobot yang ada. Skor linear ini dihitung dengan rumus $z = X \cdot w + b$, dimana X adalah fitur input. Nilai z tersebut kemudian dilewatkan ke fungsi sigmoid untuk mengubahnya menjadi probabilitas antara 0 dan 1

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

c. Menghitung *cost*

Setelah mendapatkan probabilitas, model menghitung *cost* menggunakan *cross-entropy-loss* untuk mengukur galat antara prediksi dengan nilai aktual. Jika reguralisasi seperti **l1** atau **l2** digunakan, penalti ditambahkan untuk mencegah *overfitting* dengan menambahkan penalti terhadap bobot yang besar.

d. Pembaruan parameter dengan *Gradient Descent* atau *Newton's Method*

- Jika menggunakan *Gradient Descent*, model menghitung gradien dari fungsi biaya terhadap bobot dan bias, lalu memperbarui parameter dengan mengurangkan nilai gradien yang telah dihitung, dikalikan dengan learning rate. Jika **l1** regularisasi digunakan, gradien diperbarui dengan penalti **l2**. Hal yang sama berlaku untuk **l1** regularisasi.
- Jika memilih *Newton's Method*, model menghitung Hessian (matriks turunan kedua dari *cost function*) dan menggunakan invers dari Hessian ini untuk memperbarui parameter. Namun, karena *Newton's Method* tidak kompatibel dengan **l1** regularisasi, karena **l1** regularisasi menyebabkan fungsi biaya menjadi tidak halus dan tidak diferensial pada beberapa titik, kode akan memberikan error jika **l1** regularisasi digunakan dengan metode ini.

e. Menghindari masalah numerik

Selama iterasi, model memeriksa apakah nilai probabilitas yang dihitung mendekati 0 atau 1, karena ini bisa menyebabkan matriks Hessian menjadi tidak stabil (*ill-conditioned*). Ketika ini terjadi, model memberikan peringatan untuk menunjukkan potensi masalah.

f. Iterasi dan konvergensi

Proses pelatihan terus berlanjut selama jumlah iterasi yang ditentukan atau hingga perubahan pada fungsi biaya sangat kecil (menandakan konvergensi). Setiap iterasi bertujuan untuk mengurangi fungsi biaya dan mendekati parameter optimal.

g. Prediksi kelas

Setelah pelatihan selesai, model siap digunakan untuk memprediksi kelas pada data baru (X_test). Probabilitas dihitung dengan cara yang sama seperti saat pelatihan, dan keputusan kelas dibuat berdasarkan threshold (0.5). Jika probabilitas di atas *threshold*, sampel diklasifikasikan sebagai kelas positif, jika di bawah, sebagai negatif.

4. Perbandingan hasil evaluasi model

Berdasarkan hasil evaluasi model, terdapat perbedaan kecil antara Logistic Regression yang diimplementasikan dari *scratch* dengan Logistic Regression dari scikit-learn. Akurasi model scikit-learn sedikit lebih tinggi (0.81 vs 0.80), dan F1 Score model scikit-learn juga lebih tinggi (0.60 vs 0.55). Ada beberapa kemungkinan yang menyebabkan perbedaan tersebut :

- Optimisasi yang lebih baik di scikit-learn
  Kemungkinan Scikit-learn menggunakan solver yang lebih efisien seperti liblinear atau lbfgs dan teroptimalkan untuk berbagai skenario. Sementara itu, implementasi saya hanya menggunakan *gradient descent* dan *newton's method* yang mungkin kurang optimal tanpa penyetelan parameter tambahan.
- Regularisasi dan stabilitas numerik
  Dalam kode scratch, *Newton's Method* tidak mendukung **l1** regularisasi karena perhitungan Hessian yang menjadi kompleks dan tidak stabil dengan penalti **l1**. Oleh karena itu, kode melakukan raise error jika **l1** regularisasi dipilih bersama dengan metode Newton, mengarahkan pengguna untuk menggunakan *Gradient Descent* atau memilih **l2** regularisasi. Hal ini menunjukkan batasan dalam fleksibilitas model dari scratch dibandingkan dengan model scikit-learn yang menangani regularisasi lebih baik.
- Penanganan numerik dan perhitungan gradien
  Kemungkinan, model scikit-learn dirancang dengan penanganan numerik yang lebih stabil, menghindari masalah seperti overflow atau underflow saat menghitung sigmoid. Implementasi saya menggunakan np.clip pada input sigmoid sebagai langkah pencegahan, tetapi scikit-learn memiliki metode penanganan masalah numerik yang lebih baik.

5. Improvement yang bisa dilakukan

Berikut adalah beberapa ide improvisasi yang mungkin bisa dilakukan :

a. Penyetelan *learning rate* yang optimal menggunakan *grid search*
b. Metode optimasi yang lebih efisien

   Selain *Gradient Descent* yang digunakan saat ini, mempertimbangkan metode optimasi lain seperti *Stochastic Gradient Descent* (SGD), *Mini-batch Gradient Descent*, atau *Adam Optimizer* juga mungkin bisa menjadi opsi yang menjanjikan dalam konteks improvisasi karena metode-metode tersebut memiliki kecepatan konvergen dan stabilitas yang lebih baik.

c. Implementasi regularisasi yang lebih fleksibel

Saat ini ada pembatasan dalam penggunaan **l1** regularisasi dengan *Newton's Method* karena masalah stabilitas numerik dan kompleksitas Hessian. Oleh karena itu, menambahkan dukungan untuk **l1** dengan pendekatan lain seperti *Proximal Gradient Descent* atau *Coordinate Descent* mungkin juga bisa meningkatkan fleksibilitas model dan performa pada data yang lebih bervariasi.

d. Menggunakan teknik penanganan numerik yang lebih baik dari pada np.clip
e. Peningkatan atau peningkatan metode newton