Algorithms

Jimmy Zhan

August 2025

Explanations

$$F(\omega) = \sum_{i=1}^{n} l(z_i), \quad z_i = \alpha + \beta^\intercal x_i, \tag{1}$$

where

$$\omega = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$F(\omega)$ is the loss function, which is the function I am aiming to optimise. In machine learning, the goal is to find a set of coefficients so that the loss function returns a minimal value.

ω is a vector that contains all the coefficients in the model. In this specific case (logistic regression), ω includes the bias term α and all the weights β.

$l(z_i)$ is the loss for a single sample. For example, in logistic regression, l is the negative log-likelihood loss function.

What equation 1 means is that the overall loss is the sum of all the loss for each sample.

z_i is the result of the ith sample from the model. It is the linear combination of features x_i and coefficients β, plus the bias α.

In logistic regression, we can assume that the samples are independent of each other. Therefore the overall loss of the whole dataset is the multiple of loss of each sample, in the log space becomes to sum.

Compactification

$$\theta(z) = \pi \cdot \frac{2(z-L)}{U-L} - \pi \tag{2}$$

The compactification process works as follows:

The Fourier series

$$f(\theta) = a_0 + \sum_{n=1}^{\infty}(a_m cos(m\theta) + b_m sin(m\theta)) \tag{3}$$

has a domain of $[-\pi, \pi]$, while $l(z)$ has a domain of $\mathbb{R}$.

So what the equation does is to ”stretch” or ”sqeeze” an interval $[L, U]$ (where the minimal z is likely to exist) to the interval $[-\pi, \pi]$.

But how?

Consider the interval $[L, U]$. If we manually move this interval so that the starting point is the origin, then it becomes $[0, U - L]$.

Then we compact this interval so that the length is 1. We only need a percentage information of z in the interval, that is, $\frac{z-L}{U-L}$.

We wish the target interval to be $[-\pi, \pi]$. It has a length of 2π. So we multiply it by 2π, to be $\frac{2\pi(z-L)}{U-L}$.

Finally, we minus π to make the interval $[-\pi, \pi]$.

Smooth compactifier

Denote $\theta = \theta(z)$. We approximate $l(z)$ by

$$l_F(z) = c_0 + \sum_{m=1}^{M}(a_m cos(m\theta(z)) + b_m sin(m\theta(z))) \tag{4}$$

The loss function $l(z)$ has a general form of

$$l(z) = -[y \cdot log(\hat{y}) + (1-y) \cdot log(1-\hat{y})] \tag{5}$$

We now use a truncated Fourier series to approximate the loss function.

$l_F(z)$ is the approximation function.

c_0, a_m, b_m are the Fourier coefficients. A set of constants calculated prior.

M is the number of terms calculated in the Fourier series.

Coefficients a_m, b_m, c_m are computed once (numerically) by sampling $l \circ \theta^{-1}$ on a dense grid of θ and doing a real FFT / discrete Fourier transform. (If you use the arctan compactifier include Jacobian when transforming integrals; for coefficient estimation you only need sampled values.)

Closed-form

The Fourier approximation gives closed-form expressions for the derivative and second derivative of l_F with respect to z. Use chain rule via $\theta = \theta(z)$

Define

$$S_1(\theta) = \sum_{m=1}^{M} m(-a_m sin(m\theta) + b_m cos(m\theta)) \tag{6}$$

and

$$S_2(\theta) = -\sum_{m=1}^{M} m^2(a_m cos(m\theta) + b_m sin(m\theta)) \tag{7}$$

Then

$$\begin{aligned} l'_F(z) &= \theta'(z) S_1(\theta(z)), \\ l''_F(z) &= \theta''(z) S_1(\theta(z)) + [\theta'(z)]^2 S_2(\theta(z)) \end{aligned} \tag{8}$$

Now for the full parameter vector $\omega = [\alpha; \beta], z_i = \omega^\intercal \tilde{x}_i$ where $\tilde{x}_i = [1; x_i]$. So the approximated objective

$$F_F(\omega) = \sum_{i=1}^{n} l_F(z_i) \tag{9}$$

has

$$\nabla F_F(\omega) = \sum_{i=1}^{n} l'_F(z_i)\tilde{x}_i \tag{10}$$

and the Hessian

$$H_F(\omega) = \sum_{i=1}^{n} l''_F(z_i)\tilde{x}_i\tilde{x}_i^{\intercal} \tag{11}$$

Newton-Fourier update:

$$\omega_{k+1} = \omega_k - (H_F(\omega_k) + \lambda I)^{-1}\nabla F_F(\omega_k) \tag{12}$$

optionally adding damping $\lambda > 0$ (Levenberg-Marquardt style) for stability.

Practical Implementation and Parameter Selection

The performance of the Newton-Fourier method depends on several key parameters.

1. Compactification Parameters $[L, U] or z_0, \lambda$:

These parameters should be chosen to capture the "active" region of the loss function. A robust approach is to perform a pre-analysis on the dataset. For instance, one could train a simple model (e.g., a few steps of gradient descent) to get a rough estimate of the distribution of z_i values and set the interval $[L, U]$ to cover, for example, three standard deviations around the mean. For the arctan mapping, z_0 can be set to the mean of z_i and λ can be chosen such that the interval of interest is mapped to a significant portion of $[\pi, \pi]$. Importantly, these parameters should be fixed before optimization begins to avoid the costly re-computation of Fourier coefficients at each iteration.

2. Number of Fourier Terms M:

The choice of M represents a trade-off between approximation accuracy and computational cost.

Small M: Leads to a coarse approximation but faster computations per iteration.

Large M: Provides a more accurate approximation of the loss function but increases the cost of evaluating the gradient and Hessian.

Selection Strategy: One can analyze the decay of the computed Fourier coefficients $|a_m|$ and $|b_m|$. Typically, these decay rapidly for smooth functions. M can be chosen such that coefficients beyond this point are negligible (e.g., below a certain machine-epsilon tolerance). Alternatively, M can be treated as a hyperparameter and tuned using a validation set

Computational Cost Analysis

The main overhead of this method lies in the pre-computation of Fourier coefficients and the per-iteration cost of evaluating the series.

For Preprocessing cost, the Fourier coefficients are computed once by sampling the loss function at K points and running an FFT. The complexity is $O(K \quad logK)$, where K is typically chosen to be much larger than M. This is a one-time, fixed cost.

Pre-Iteration Cost

Let n be the number of samples and d be the number of features.

For each of the n samples, we evaluate series S_1 with M terms, which is an $O(M)$ operation. This is scaled and summed up for d features, therefore $O(n \cdot d \cdot M)$.

For each sample, we evaluate S_1 and $S_2(O(M))$ and then form a $d \times d$ outer product, costing $O(d^2)$. Therefore $O(n \cdot d^2 \cdot M)$ in total.

Matrix Inversion: $O(d^3)$ for solving the linear system.

The Newton-Fourier method is computationally more expensive per iteration than the standard Newton method by a factor of M. Its potential advantage must therefore come from a significant reduction in the total number of iterations required for convergence.

Core Advantage: Global Approximation

The fundamental distinction of this method is its use of a global approximation.

A second-order Taylor expansion is a purely local approximation. It models the function as a parabola that is accurate only in an infinitesimal neighborhood of the current point ω_k. If the function's true shape deviates significantly from this parabola (which is common far from the optimum), the resulting Newton step can be too large, too small, or even point in a wrong direction, requiring line searches or trust regions to stabilize.

A truncated Fourier series provides a global approximation across the entire compactified interval. By capturing the overall shape, including non-quadratic features of the loss function, the resulting Hessian H_F can provide a more accurate model of the function's curvature over a much wider range. This could lead to significantly better and more direct update steps, especially during the early stages of optimization when the iterates are far from the solution. This improved global perspective has the potential to drastically reduce the number of iterations needed to reach convergence.

Idea 1 Ver 2.0

The Newton-Fourier method has the following iteration formula

$$\omega_{k+1} = \omega_k - (H_F(\omega_k) + \lambda I)^{-1} \nabla F_F(\omega_k) \tag{13}$$