Multi-Class Logistic Regression with Optimization methods

Alessio Chen
alessio.chen@edu.unifi.it

May 15, 2024

Abstract

In this assignment, we solve the Multi-Class Logistic Regression problem using two Optimization methods techniques: a line search method (Gradient descent) and a Trust region method with the Cauchy step.

1 Summary

This assignment provides an overview of optimization techniques, specifically gradient descent with Armijo line search and the trust region algorithm with the Cauchy step, applied to the Multi-class Logistic Regression problem. By conducting experiments with real-world multi-class classification datasets, we demonstrate the effectiveness of these methods, observing the convergence of the loss.

2 Multi-class logistic Regression

Multi-class logistic regression can be conceptualized as an optimization problem aimed at finding the optimal parameters w that best fit the observed data and enable accurate classification of instances into multiple classes. At its core, the task involves estimating the probability distribution over the various classes given the input features. This is achieved by minimizing a suitable loss function, which quantifies the disparity between the predicted probabilities and the true class labels. So the problem can we write as

$$\min_{w} \quad L(x, y; w) + \lambda ||w||^2 \tag{1}$$

- x , and y represent the input data and corresponding labels, respectively.
- `w` is the parameter vector to be optimized,
- `loss(x,y;w)` is the loss function measuring the discrepancy between the predicted values $\hat{y} = w^T x$,
- λ is the regularization parameter that controls the importance of the l_2 regularization term $||w||^2$.

In this assignment, we employ the `cross entropy loss`, the formulation of which is detailed in [1]. Here, we present the mathematical expression as:

$$L(x, y; w) = -\log \frac{e^{w_c^T x}}{\sum_{k=1}^{K} e^{w_k^T x}} \tag{2}$$

3 Gradient descend

Gradient descent, as a first-order optimization algorithm, iteratively computes the gradient of the loss function with respect to the parameters. Then, guided by the Armijo line search algorithm, it adjusts the parameters in the direction opposite to the gradient.

Algorithm 1: Gradient descent

Initialize w_0, λ, max_iter, $k = 0$
while $k < max\_iter$ **do**
  $d_k = -\nabla L(w_k)$
  α_k calculated with armijo
  $w_{k+1} = w_k + \alpha_k d_k$
  $k = k + 1$
end

Algorithm 2: Armijo line search

- $\nabla L(w_k)^T d_k < 0$
- $\gamma \in (0, 1)$
- $\delta \in (0, 1)$
- $\alpha_0 > 0$

$\alpha = \alpha_0, t = 0$
while $L(w_k + \alpha_t d_k) > L(w_k) + \gamma \alpha_t \nabla L(w_k)^T d_k$ **do**
  $\alpha_{t+1} = \delta \alpha_t$
  $t = t + 1$
end

The formulation of $\nabla L(x, y; w)$ it's detailed in [2] expressed as:

$$\nabla L(w) = \frac{1}{N} \sum_{n=1}^{N} (y_n - p_n) \otimes x \tag{3}$$

- `N` is the number of samples.
- y_n represents the true label of the n-th training sample.
- p_n represents the predicted probability vector for all classes obtained from the logistic regression model.
- `x` represents the input feature vector for the n-th training sample.

4 Trust region

Trust region methods are optimization algorithms that minimize an objective function while incorporating a trust region around the current solution. Within this region, they construct a model of the objective function and iteratively update the solution by computing steps that minimize the model within the trust region. This approach provides a more robust alternative to line search methods, offering stability and convergence guarantees in various optimization scenarios. Trust region methods are particularly effective when dealing with non-convex, ill-conditioned, or noisy objective functions, making them a valuable tool in the optimization toolbox.
The algorithm can be summarized with the following pseudo-code:

Algorithm 3: Trust Region Method

Initialize $\eta_1, \eta_2, \gamma_1, \gamma_2, w_0$, trust radius Δ_0, maximum trust radius $\Delta_{\max}$, max_iter
$0 < \eta_1 < \eta_2 < 1$
$0 < \gamma_1 < \gamma_2 < 1$
$\Delta_0 > 0$
k = 0
while $k < max\_iter$ **do**
  Construct a quadratic model of loss function within trust region 4.1
  Solve sub-problem to obtain step s_k 4.2
  Acceptance of the test point 4.3
  Adjust trust radius 4.4
  $k = k + 1$
end

In the following sub-sections we are reporting some of the key points of the algorithm, for a detailed explanation of the trust region method and its implementation, refer to ([3], Ch 9.1).

4.1 Quadratic model of loss function

At a generic iteration k of the algorithm aimed at approximating the loss function, a quadratic model $m_k(s)$ is employed. This model is characterized by the following expression:

$$m_k(s) = L(w_k) + \nabla f(w_k)^T s + \frac{1}{2} s^T B_k s \tag{4}$$

- w_k represents the current parameter vector at iteration k.
- B_k in general, it is a symmetric matrix. In our specific case, it corresponds to the Hessian $\nabla^2 L(w_k)$, as detailed in [2],

The formulation of the Hessian matrix is elaborated as follows:

$$\nabla L(w)^2 = \frac{1}{N} \sum_{n=1}^{N} (D(p_n) - p_n p_n^T) \otimes x_n x_n^T \tag{5}$$

- p_n represents the predicted probability vector for the n-th sample.
- x_n denotes the feature vector for the n-th sample.
- $D(p_n)$ refers to the diagonal matrix, where each diagonal element is computed as $p_{ni}(1-p_{ni})$ where p_{ni} is the probability assigned to class i by the model.

4.2 Sub-problem Solution - Cauchy Step

For this assignment, the `Cauchy step` is employed to determine an approximated solution s_k that produces a sufficient decrement $m_k(s_k) - m_k(0)$ of the quadratic model sub-problem.

$$s_k = -\tau^* \nabla \mathrm{L}(w_k) \tag{6}$$

$$\tau^* = \begin{cases} \frac{\nabla L(w_k)}{||\nabla L(w_k)||} & \nabla L(w_k)^T B_k \nabla L(w_k) \leq 0 \\ \min\{\frac{\nabla L(w_K)}{||\nabla L(w_k)||}, \frac{||\nabla L(w_k)||^2}{\nabla L(w_k)^T B_k \nabla L(w_k)}\} & \nabla L(w_k)^T B_k \nabla L(w_k) > 0 \end{cases} \tag{7}$$

4.3 Acceptance of the test point

Compute $L(w_k + s_k)$ and set:

$$\rho_k = \frac{L(w_k) - L(w_k + s_k)}{m_k(0) - m_k(s_k)} \tag{8}$$

$$w_{k+1} = \begin{cases} w_k + s_k & \rho_k \geq \eta_1 \\ w_k & \text{otherwise} \end{cases} \tag{9}$$

4.4 Adjust trust radius

The new trust radius Δ_{k+1} is defined as:

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty) & \rho_k \geq \eta_2 \\ [\gamma_2 \Delta_k, \Delta_k] & \rho_k \in [\eta_1, \eta_2] \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \rho_k < \eta_1 \end{cases} \tag{10}$$

5 Experiments

The efficiency of the considered methods is assessed by measuring the decrease in training loss over successive iterations. Additionally, the accuracy of the models trained on each real-world dataset is reported for both optimization methods. These accuracies are then compared to those obtained using the `scikit-learn` library.

5.1 Iris Data set

The Iris dataset is a classic and straightforward multi-class classification dataset commonly used in machine learning.

- Classes: 3
- Features: 4
- Sample per class: 50
- Samples total: 150

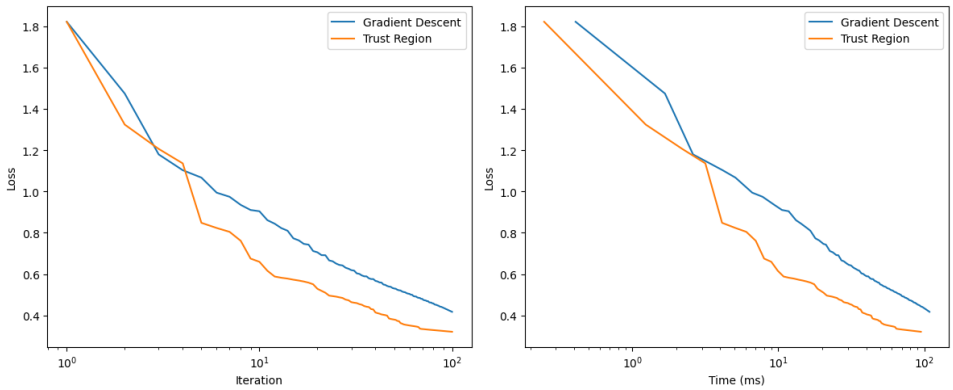

Figure 1: Development of loss during Training for Iris dataset

5.2 Digits data set

The Digit dataset is made up of 1797 8x8 images. Each image is of a hand-written digit.

- Classes: 10
- Features: 64
- Sample per class: 180
- Samples total: 1797

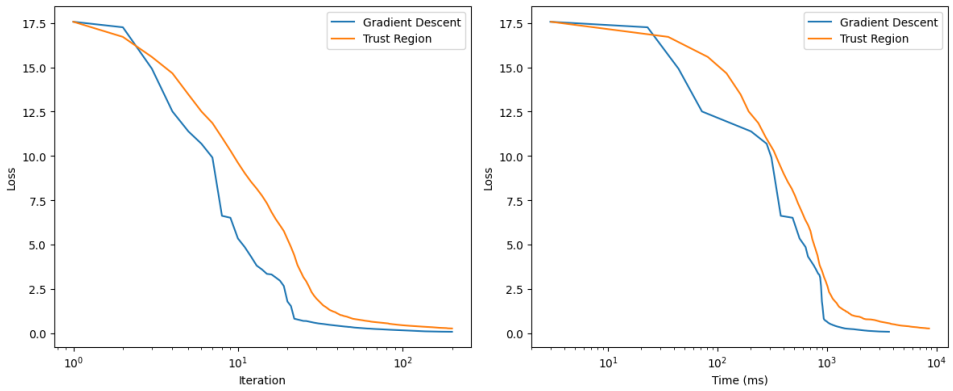

Figure 2: Development of loss during Training for Digits dataset

5.3 Dna data set

The dataset originates from the StatLog project and consists of data points described by 180 indicator binary variables. The objective is to classify the data into one of three classes: "ei", "ie", or "neither", which correspond to the boundaries between exons (the retained parts of DNA sequences after splicing) and introns (the spliced-out parts of DNA sequences).

- Classes: 10
- Features: 180
- Samples total: 3186

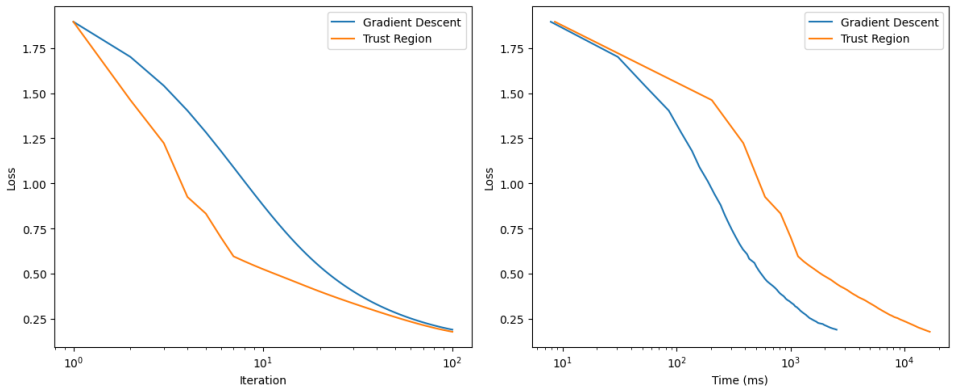

Figure 3: Development of loss during Training for Dna dataset

5.4 Segment data set

The Segment dataset is an image segmentation database. The instances were drawn randomly from a database of 7 outdoor images. The images were handsegmented to create a classification for every pixel. Each instance is a 3x3 region.

- Classes: 7
- Features: 19
- Samples total: 2360

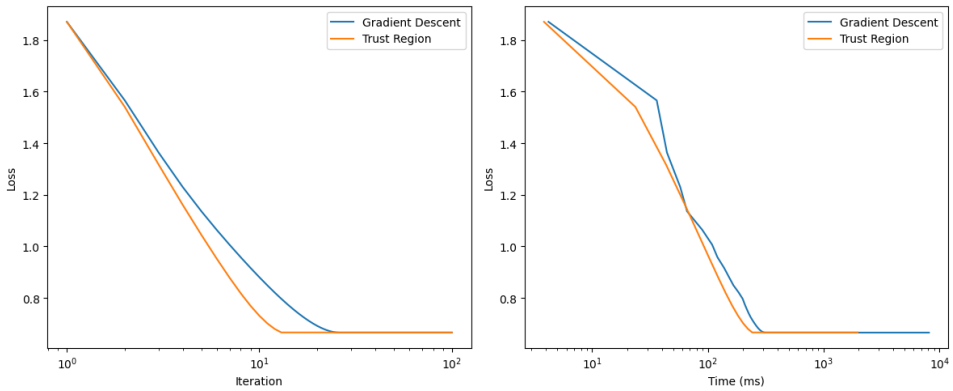

Figure 4: Development of loss during Training for Segment dataset

5.5 Accuracies

Table 1 presents the accuracies obtained through fine-tuning parameters for two algorithms, compared with those of the scikit-learn implementations utilizing lbfgs with l2 regularization.

In our optimization, we applied a regularization parameter of $\lambda = 10^{-2}$ within our loss function. Correspondingly, scikit-learn employs a parameter $C = \frac{1}{\lambda}$. We experimented with fitting the model using both the default $C = 1$ and C derived from $\lambda = 10^{-2}$. Interestingly, despite minimal alterations in accuracy, there were notable discrepancies in the norms of weights $|w|$.

Data set	**Gradient descend**	**Trust region**	**scikit-learn C=1**	**scikit-learn** $C = 10^2$
Iris	0.96	0.95	0.96	0.97
Digits	0.92	0.87	0.95	0.94
Dna	0.92	0.90	0.94	0.92
Segment	0.76	0.77	0.79	0.78

Table 1: Table of accuracies

Data set	**Gradient descend**	**Trust region**	**scikit-learn C=1**	**scikit-learn** $C = 10^2$
Iris	2.70	5.35	3.30	19.18
Digits	8.13	12.27	2.78	6.30
Dna	7.70	6.95	10.50	44.68
Segment	6.53	6.70	9.18	119.16

Table 2: Table for $|w|$

References

[1] S. Ji, Y. Xie. Logistic Regression: From Binary to Multi-Class

[2] T. Vu. (2016). Multinomial Logistic Regression: Convexity and Smoothness

[3] Grippo, L., Sciandrone, M. (2011). "Metodi di ottimizzazione non vincolata", Springer.

[4] LIBSVM Data: Classification, Regression, and Multi-label

[5] Scikit-learn

[6] Github code