# Optimization applied on a real dataset

Silviu Filote 1059252

University of Bergamo
Engineering department

November 7, 2024

# Introduction

The aim of this project is to implement and analyze various optimization algorithms, studied during the optimization course on a real-world dataset. The dataset, sourced from **kaggle**, focuses on the classification of banana quality, where features such as softness, size, and other measurable attributes are used to predict whether the bananas are of good or bad quality.

By applying different optimization techniques to this classification task, we will evaluate their performance in terms of convergence rate, computational efficiency, and accuracy. The key performance metrics will include the number of iterations (steps) required for convergence, execution time, accuracy of the classification, and the overall computational complexity of each algorithm. The final analysis will provide insights into the effectiveness of these optimization approaches in handling real-world classification problems.

# Dataset analysis

- The dataset, titled "Banana Quality", was sourced from Kaggle and consists of 8.000 complete records, with no missing data.
- Key features in the dataset include: *Size*, *Weight*, *Sweetness*, *Softness*, *Harvest Time*, *Ripeness*, *Acidity*, and *Quality*. These features provide a comprehensive representation of banana attributes relevant to their classification.
- The Quality feature, initially categorical (e.g., "Good" or "Bad"), was converted into a numerical format for model training and analysis, enabling seamless integration into various optimization and machine learning algorithms.
- Following the Exploratory Data Analysis (EDA), the preprocessing steps will include examining the distribution of the quality labels, removing outliers using the IQR method, and applying normalization and standardization to the relevant features to ensure they are on comparable scales.

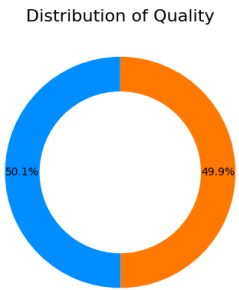


**Figure 1:** Shows that the distribution of the quality feature is nearly perfectly balanced. We should expect the classifiers to perform effectively, without bias toward any specific class.

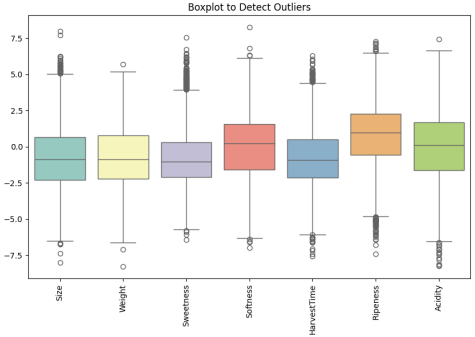


**Figure 2:** This figure presents a boxplot for each feature in the dataset, providing a clear visualization of potential outliers. The boxplot helps to identify extreme values that fall outside the interquartile range (IQR), represented by points beyond the whiskers, which may indicate anomalies or data points that deviate significantly from the majority of the observations.

# Maximum Likelihood Estimation (MLE)

- MLE is a statistical method for estimating the parameters of a probabilistic model. It finds the parameter values that make the observed data most likely, given a chosen probability distribution
- The joint probability density function (pdf) of the observed data is interpreted as the likelihood function when viewed as a function of the model parameters
- MLE seeks to maximize the likelihood function $\mathcal{L}(\boldsymbol{\theta}|\, Y)$, or equivalently, its logarithm $\ln \mathcal{L}(\boldsymbol{\theta}|\, Y)$, to obtain the parameter estimates $\boldsymbol{\theta}_{ML}$
- **Key properties:** asymptotic correctness, consistency, asymptotic efficiency and asymptotic normality
- Computationally more efficient to maximaze the log-likelihood
- Minimizing the negative log-likelihood is equivalent to maximizing the log-likelihood getting a min problem
- Maximizing the likelihood or log-likelihood requires **iterative optimization methods**

$$
\begin{aligned}
f_Y(y(1), y(2), \ldots, y(N)|\mu, \sigma^2) &= f_Y(\,Y|\mu, \sigma^2) \\
&= \mathcal{L}(\mu, \sigma^2|\,Y) \\
&= \prod_{i=1}^{N} \mathcal{N}(y(i)|\mu, \sigma^2) \\
&= f_y(y(1)|\mu, \sigma^2) \cdot \ldots \cdot f_y(y(N)|\mu, \sigma^2)
\end{aligned}
\tag{1}
$$

$$
\underset{2\times 1}{\widehat{\boldsymbol{\theta}}_{ML}} = \begin{bmatrix} \widehat{\mu} \\ \widehat{\sigma}^2 \end{bmatrix} = \arg\max_{\boldsymbol{\theta}} \; \mathcal{L}(\boldsymbol{\theta}|\,Y) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{N} \mathcal{N}\big(y(i)|\mu, \sigma^2\big) \tag{2}
$$

**Figure 3:** this figure illustrates the maximum likelihood estimation (MLE) for a normal distribution. The likelihood function $\mathcal{L}(\mu, \sigma^2|\,Y)$ is the product of individual normal probability densities for each data point $y(i)$, given parameters $\mu$ (mean) and $\sigma^2$ (variance). The MLE finds the parameter values $\widehat{\mu}$ and $\widehat{\sigma}^2$ that maximize this likelihood function, representing the most likely parameters given the data.

| | |
|---|---|
| **Asymptotically correct:** | $\lim_{N \to +\infty} \mathbb{E}\left[\widehat{\boldsymbol{\theta}}_{ML}\right] = \boldsymbol{\theta}^0$ |
| **Consistent:** | the larger the $N$, the more precise the estimate |
| **Asymptotically efficient:** | $\lim_{N \to +\infty} \mathrm{Var}\left[\widehat{\boldsymbol{\theta}}_{ML}\right] = M^{-1}$ |
| **Asymptotically normal:** | $\widehat{\boldsymbol{\theta}}_{ML} \sim \mathcal{N}(\boldsymbol{\theta}^0, M^{-1}) \quad \text{as } N \to +\infty$ |

**Figure 4:** these properties demonstrate the robustness and reliability of MLE in statistical estimation.

# Methodology

- We will implement **logistic regression**, which models the probability of a binary outcome as a linear combination of the datasets features. The objective function will be set using the **log-likelihood**, which quantifies how likely the observed data is given the models parameters. This method assumes that the datasets observations follow a specific **probability distribution** (typically a Bernoulli distribution for binary classification tasks).
- To ensure that the **objective function** (log-likelihood) is appropriate for optimization, we will apply relevant **optimization theorems**. This involves checking the function's **convexity**, **differentiability**, and **smoothness**, which are essential properties that ensure convergence to a global minimum during optimization.
- We will test different algorithms and techniques applied to see how they behave on the selected dataset. Those algorithms are: gradient descent (GD), stochastic gradient descent (SGD), mini-batch gradient descent, accelerated gradient descent (Nesterov), and Newton's method.

We are given a dataset $\mathcal{D} = \{\{\mathbf{x}(1), y(1)\}, \ldots, \{\mathbf{x}(n), y(n)\}\}$, where each feature vector $\mathbf{x}(i) \in \mathbb{R}^{(d\times 1)}$ corresponds to an observation, and $y(i)$ is a categorical variable representing a binary class, either 0 or 1. The observations are independent and identically distributed (iid), with $N$ total data points.

To model the probability of class membership, we define a linear combination of the features:

$$a = \sum_{j=0}^{d-1} x_j \cdot \theta_j = \boldsymbol{x}^\top \cdot \boldsymbol{\theta} \in \mathbb{R} \tag{3}$$

Let $\boldsymbol{\theta}$ be the vector of parameters to be estimated. This linear combination $a$ represents the weighted sum of the features. The logistic function, or sigmoid function, is then applied to $a$ to map the output to the interval $[0, 1]$, providing a probability interpretation:

$$s(a) = \frac{1}{1+e^{-a}} = \frac{e^a}{a+e^a} = \begin{cases} a \gg 0 & \Rightarrow \quad s(a) \approx 1 \\ a \ll 0 & \Rightarrow \quad s(a) \approx 0 \end{cases} \tag{4}$$

The sigmoid function ensures that the output is constrained between 0 and 1, making it suitable for modeling probabilities. This means that, depending on the values of the features and the parameters $\boldsymbol{\theta}$, we can estimate the likelihood that an observation belongs to class 1 or class 0.

In the context of logistic regression, the model estimates the probability that $y = 1$ using the following formula:

$$P(y = 1|\boldsymbol{x}) = s(a) = s(\boldsymbol{x}^T \cdot \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}^T \cdot \boldsymbol{\theta}}} \equiv \pi \tag{5}$$

The probability $\pi$ is the models estimate for the likelihood that the observation $\mathbf{x}$ belongs to class 1. The higher the value of $\pi$, the more confident the model is that the observation belongs to class 1. Otherwise, if $\pi$ is close to 0, the model predicts that the observation is likely to belong to class 0. Logistic regression assumes that the binary response variable $y$ follows a Bernoulli distribution:

$$y \sim \text{Bernoulli}(\pi) = \pi^y \cdot (1 - \pi)^{1-y} \tag{6}$$

This formulation provides a probabilistic framework, where each observation either takes the value $y = 1$ with probability $\pi$ or $y = 0$ with probability $1 - \pi$. To fit the logistic regression model, we seek to maximize the likelihood function, which represents the probability of observing the given data as a function of the model parameters:

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \prod_{i=1}^{N} P\big(y(i) = 1|\boldsymbol{\varphi(i)}, \boldsymbol{\theta}\big) = \prod_{i=1}^{N} \pi(i)^{y(i)} \cdot \big(1 - \pi(i)\big)^{1-y(i)} \tag{7}$$

The goal is to maximize the likelihood function, but instead, we typically maximize the log-likelihood function due to its simpler mathematical properties. By applying the logarithm, we convert the product into a sum, which is easier to handle in optimization algorithms. In the context of logistic regression, this involves finding the parameters $\boldsymbol{\theta}$ that minimize the negative log-likelihood function, which is equivalent to maximizing the likelihood:

$$
\begin{aligned}
-\ln\left[\mathcal{L}(\pi|Y)\right] &= -\ln\left(\prod_{i=1}^{N}\pi(i)^{y(i)}\cdot(1-\pi(i))^{1-y(i)}\right)\\
&= -\sum_{i=1}^{N}\ln\left(\pi(i)^{y(i)}\cdot(1-\pi(i))^{1-y(i)}\right)\\
&= -\sum_{i=1}^{N}\left(\ln\left[\pi(i)^{y(i)}\right]+\ln\left[(1-\pi(i))^{1-y(i)}\right]\right)\\
&= -\sum_{i=1}^{N}\left(y(i)\cdot\ln\left[\pi(i)\right]+(1-y(i))\cdot\ln\left[1-\pi(i)\right]\right)\\
&= -\sum_{i=1}^{N}\left(y(i)\cdot\ln\left[\frac{1}{1+e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\right]+(1-y(i))\cdot\ln\left[\frac{e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}{1+e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\right]\right)\\
&= -\sum_{i=1}^{N}\left(y(i)\cdot\ln\left[\frac{1}{1+e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\cdot\frac{1+e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}{e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\right]+\cdot\ln\left[\frac{e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}{1+e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\right]\right)\\
&= -\sum_{i=1}^{N}\left(y(i)\cdot\ln\left[\frac{1}{e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\right]+\ln\left[\frac{e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}{1+e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\right]\right)\\
&= -\sum_{i=1}^{N}\left(y(i)\cdot\boldsymbol{x}_i^\top\boldsymbol{\theta}+\ln\left[\frac{e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}{1+e^{-\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\right]\right)\\
&= -\sum_{i=1}^{N}\left(y(i)\cdot\boldsymbol{x}_i^\top\boldsymbol{\theta}+\ln\left[\frac{1}{1+e^{\boldsymbol{x}_i^\top\boldsymbol{\theta}}}\right]\right)\\
&= -\sum_{i=1}^{N}\left(y(i)\cdot\boldsymbol{x}_i^\top\boldsymbol{\theta}-\ln\left[1+e^{\boldsymbol{x}_i^\top\boldsymbol{\theta}}\right]\right)\equiv J(\boldsymbol{\theta})
\end{aligned}
\tag{8}
$$

The (8) is known as the cost function or objective function for logistic regression, and the aim is to find the parameters $\hat{\boldsymbol{\theta}}$ that minimize this objective function $\mathcal{J}(\boldsymbol{\theta})$:

$$\hat{\theta} = \arg\min_{\theta} J(\theta) = \arg\min_{\theta} \left\{ -\sum_{i=1}^{N} \left[ \ln\left(1 + e^{\boldsymbol{x}_i^\top \theta}\right) - y(i) \cdot \boldsymbol{x}_i^\top \theta \right] \right\} \tag{9}$$

We are also interested in calculating the gradient and the Hessian of the objective function. To obtain the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$, we differentiate $\mathcal{J}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, where the derivative of $\ln(1 + e^{-\mathbf{x}_i^T \boldsymbol{\theta}})$ is (10) and therefore, the gradient becomes (11):

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ln(1 + e^{\boldsymbol{x}_i^\top \boldsymbol{\theta}}) = \frac{e^{\boldsymbol{x}_i^\top \boldsymbol{\theta}}}{1 + e^{\boldsymbol{x}_i^\top \boldsymbol{\theta}}} \cdot \boldsymbol{x}_i = \pi(i) \cdot \boldsymbol{x}_i \tag{10}$$

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= -\sum_{i=1}^{N} \left( y(i) \cdot \boldsymbol{x}_i - \pi(i) \cdot \boldsymbol{x}_i \right) \\ &= -\sum_{i=1}^{N} \left( y(i) - \pi(i) \right) \cdot \boldsymbol{x}_i \end{aligned} \tag{11}$$

The gradient or score function can be written in matrix form, if we define:

- $\mathbf{X}$ as an $N \times d$ matrix of input features, where each row is $\mathbf{x}_i^T$
- $\mathbf{y}$ as the vector of target values $y(i)$
- $\hat{\mathbf{y}} \equiv \boldsymbol{\pi} = \frac{1}{1+e^{-\mathbf{x}_i^T \boldsymbol{\theta}}}$ as the vector of predicted probabilities

$$\begin{aligned} \nabla_\theta J(\theta) &= -\mathbf{X}^\top (\boldsymbol{y} - \boldsymbol{\pi}) \\ &= -\mathbf{X}^\top (\boldsymbol{y} - \widehat{y}) \end{aligned} \tag{12}$$

Following the same approach, to compute the Hessian matrix $\mathbf{H}(\boldsymbol{\theta}) = \nabla^2_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$, we differentiate the gradient with respect to $\boldsymbol{\theta}$, where the derivative of $\pi(i)$ with respect to $\boldsymbol{\theta}$ is given by (13) and so we can express the Hessian as (14):

$$\nabla_{\boldsymbol{\theta}} \pi(i) = \pi(i)(1 - \pi(i)) \cdot \boldsymbol{x}_i \tag{13}$$

$$\begin{aligned} H(\boldsymbol{\theta}) \equiv \nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \left(\pi(i) - y(i)\right) \cdot \boldsymbol{x}_i \\ &= \sum_{i=1}^{N} \pi(i)(1 - \pi(i)) \cdot \boldsymbol{x}_i \cdot \boldsymbol{x}_i^\top \end{aligned} \tag{14}$$

The Hessian can also be written in matrix form, if we define:

- $W$ as the $N \times N$ diagonal matrix, where each diagonal element is defined as $\pi(i)(1-\pi(i))$,
- $X$ as the $N \times d$ matrix of input features, assumed to be a full-rank matrix.

The score function can then be expressed as:

$$W = \begin{bmatrix} \pi(1)(1-\pi(1)) & 0 & 0 & \cdots & 0 \\ 0 & \pi(2)(1-\pi(2)) & 0 & \cdots & 0 \\ 0 & 0 & \pi(3)(1-\pi(3)) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \pi(N)(1-\pi(N)) \end{bmatrix} \quad X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_N^\top \end{bmatrix}$$

The Hessian matrix is given by:

$$H(\boldsymbol{\theta}) = X^\top W X \tag{15}$$

To prove that $J(\theta)$ is convex, we need to show that the Hessian matrix is positive semi-definite i.e., the Hessian has non-negative eigenvalues. The Hessian of $J(\theta)$ is the following:

$$H(\theta) = \nabla^2_\theta J(\theta) = \sum_{i=1}^{N} \pi(i)(1 - \pi(i)) \cdot \mathbf{x}_i \cdot \mathbf{x}_i^\top$$

This Hessian can also be expressed in matrix form as:

$$H(\theta) = \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}$$

where:

- $\boldsymbol{X}$ is the design matrix (rows are $\boldsymbol{x}_i^\top$),
- $\boldsymbol{W}$ is a diagonal matrix with entries $\pi(i)(1 - \pi(i)) \in [0, 0.25]$ since the derivative of the sigmoid function is bounded by 0.25:

$$\pi(i)(1 - \pi(i)) \leq 0.25$$

Since $\boldsymbol{W}$ is a diagonal matrix with non-negative entries, and $\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}$ is the product of real matrices, the Hessian is positive semi-definite. Thus, the function $J(\theta)$ is convex.

For the logistic regression objective function to be Lipschitz continuous, we need to show that the norm of the gradient is bounded, i.e. there exists a constant $B$ such that:

$$\|\nabla J(\theta)\| \leq B \quad \forall \theta$$

The gradient of the logistic loss is:

$$\nabla_\theta J(\theta) = \boldsymbol{X}^\top (\boldsymbol{\pi} - \boldsymbol{y})$$

Where:

- $\boldsymbol{X}$ is the design matrix (with each row representing a feature vector $\boldsymbol{x}^{(i)}$),
- $\boldsymbol{\pi}$ is the vector of predicted probabilities, which is computed using the logistic function, and
- $\boldsymbol{y}$ is the vector of true labels.

To find the Lipschitz constant $B$, we can bound the gradient as follows:

$$\|\nabla J(\theta)\| = \|\boldsymbol{X}^\top (\boldsymbol{\pi} - \boldsymbol{y})\| \leq \|\boldsymbol{X}^\top\| \cdot \|\boldsymbol{\pi} - \boldsymbol{y}\|$$

Here, we use the property of norms that states the norm of a product is less than or equal to the product of the norms. This is known as Cauchy schwarz inequality.

Now, we need to bound the term $\|\boldsymbol{\pi} - \boldsymbol{y}\|$. Since the predicted probabilities $\pi(i)$ lie in the interval $[0, 1]$, we can estimate:

$$\|\boldsymbol{\pi} - \boldsymbol{y}\| \leq \sqrt{N} \cdot \max_i |\pi(i) - y(i)|$$

Where $N$ is the number of samples. Heres the rationale:

- The difference $\pi(i) - y(i)$ can take values in $[-1, 1]$, which means that the maximum absolute difference can be 1 for any sample.
- If we assume $\max_i |\pi(i) - y(i)| = 1$ in the worst-case scenario, then the bound simplifies to $\sqrt{N}$, which reflects the relationship between the L2 and L$\infty$ norms.

Thus, the maximum possible value for $\|\boldsymbol{\pi} - \boldsymbol{y}\|$ will be at most $N$ (more understandable bounding):

$$\|\boldsymbol{\pi} - \boldsymbol{y}\| \leq N$$

Substituting this back into the inequality for the gradient, we obtain:

$$\|\nabla J(\theta)\| \leq \|\boldsymbol{X}^\top\| \cdot N$$

Thus, we can define the Lipschitz constant $B$ as:

$$B = \|\boldsymbol{X}^\top\| \cdot N$$

A function is smooth if its hessian is bounded in norm, such that:

$$\|\nabla^2 J(\theta)\| \leq L \quad \forall \theta$$

The Hessian $H(\theta)$ is:

$$H(\theta) = \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}$$

where $\boldsymbol{W} = \text{diag}(\pi^{(i)}(1 - \pi^{(i)}))$ and since $\pi(i)(1 - \pi(i)) \leq 0.25 = \frac{1}{4}$ then we have the following bound:

$$L = \|H(\theta)\|_2 = \|\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}\|_2 \leq \frac{1}{4} \|\boldsymbol{X}^\top \boldsymbol{X}\|_2$$

Here, $\|\boldsymbol{X}^\top \boldsymbol{X}\|_2$ is the spectral norm (largest eigenvalue) of the matrix $\boldsymbol{X}^\top \boldsymbol{X}$. Thus, the logistic regression objective is smooth with smoothness parameter:

$$L = \frac{1}{4} \|\boldsymbol{X}^\top \boldsymbol{X}\|_2$$

# Algorithms

**Table 1:** Overview of algorithms executed with a maximum step count of $t_{max} = 20,000$. Each algorithm terminates when the tolerance condition $f(\boldsymbol{x}) - f(\boldsymbol{x}^*) \leq \varepsilon = 10^{-6}$ is met. The dots in the iterative step column indicate that the iterative steps for these algorithms are similar to those described immediately above.

| **Algorithm** | **Iterative Step** | **Notes** |
|---|---|---|
| Gradient descent (GD0) | $\boldsymbol{x}_{t+1} := \boldsymbol{x}_t - \gamma \nabla f(\boldsymbol{x}_t)$ | $\boldsymbol{x}_0 \in \mathbb{R}^d$, *stepsize* $\gamma \geq 0$ |
| Gradient descent (GD1) | ... | Bold driver algorithm ($m$ steps) |
| Gradient descent (GD2) | ... | Armijo rule |
| Gradient descent (GD3) | ... | Binary search |
| Gradient descent (smooth) | $\boldsymbol{x}_{t+1} := \boldsymbol{x}_t - \frac{1}{L} \nabla f(\boldsymbol{x}_t)$ | $\gamma = \frac{1}{L}$, where $L = \frac{1}{4}\|\boldsymbol{X}^T \boldsymbol{X}\|$ |
| Stochastic GD (SGD1) | $\boldsymbol{x}_{t+1} := \boldsymbol{x}_t - \gamma_t \nabla f_i(\boldsymbol{x}_t)$ | $t = 0, 1, \ldots$: sample $i \in \{1, \ldots, N\}$ uniformly at random |
| Stochastic GD (SGD2) | $\boldsymbol{x}_{t+1} := \boldsymbol{x}_t - \gamma_t \frac{1}{m} \sum_{j=1}^{m} \nabla f_{h_j}(\boldsymbol{x}_t)$ | $t = 0, 1, \ldots$: random subset $\{h_1, \ldots, h_m\} \subseteq \{1, \ldots, N\}$ |
| Accelerated descent | $\boldsymbol{y}_{t+1} := \boldsymbol{x}_t - \frac{1}{L} \nabla f(\boldsymbol{x}_t)$ | $\boldsymbol{z}_0 = \boldsymbol{y}_0 = \boldsymbol{x}_0 \in \mathbb{R}^d$ |
| | $\boldsymbol{z}_{t+1} := \boldsymbol{z}_t - \frac{t+1}{2L} \nabla f(\boldsymbol{x}_t)$ | $\gamma = \frac{1}{L}$, where $L = \frac{1}{4}\|\boldsymbol{X}^T \boldsymbol{X}\|$ |
| | $\boldsymbol{x}_{t+1} := \frac{t+1}{t+3} \boldsymbol{y}_{t+1} + \frac{2}{t+3} \boldsymbol{z}_{t+1}$ | |
| Newton method | $\boldsymbol{x}_{t+1} := \boldsymbol{x}_t - H(\boldsymbol{x}_t) \nabla f(\boldsymbol{x}_t)$ | $H(\boldsymbol{x}_t) = \left(\nabla^2 f(\boldsymbol{x}_t)\right)^{-1}$ is the inverse Hessian |
| Coordinate Descent | $\boldsymbol{x}_{t+1} := \boldsymbol{x}_t - \frac{1}{L} \nabla_{i_t} f(\boldsymbol{x}_t) \boldsymbol{e}_{i_t}$ | $i_t$ is the coordinate updated at step $t$, $\boldsymbol{e}_{i_t}$ is the unit vector |

# Results

**Table 2:** Performance comparison of various gradient descent algorithms. The initial $\boldsymbol{x}_0$ values were randomized to ensure an unbiased starting point and also $\boldsymbol{x}_0$ is the same for all the algorithms. The table includes the number of steps taken and the execution time for each algorithm, highlighting the efficiency and effectiveness of each approach in minimizing the logistic loss function.

| **Algorithm** | **Steps** | **Execution Time (s)** |
|---|---|---|
| Gradient descent (GD0) | 406 | 0.3743 |
| Gradient descent (GD1) | 1 | 0.0103 |
| Gradient descent (GD2) | 46 | 0.4097 |
| Gradient descent (GD3) | 36 | 1.8741 |
| Gradient descent smooth | 2 | 0.0037 |
| Stochastic GD (SGD1) | 9 | 0.0098 |
| Stochastic GD (SGD2) | 1994 | 2.0358 |
| Accelerated descent | 23 | 0.0339 |
| Newton method | 2 | 0.5730 |
| Coordinate Descent | 84 | 0.1078 |

**Estimated $\hat{\boldsymbol{x}}$ vectors:**

$$\begin{bmatrix} -0.276 \\ 1.400 \\ 1.988 \\ 1.373 \\ 0.176 \\ 1.115 \\ 1.298 \\ -0.238 \end{bmatrix} \begin{bmatrix} -0.276 \\ 1.407 \\ 1.996 \\ 1.376 \\ 0.175 \\ 1.118 \\ 1.296 \\ -0.245 \end{bmatrix} \begin{bmatrix} -0.276 \\ 1.400 \\ 1.988 \\ 1.373 \\ 0.176 \\ 1.115 \\ 1.298 \\ -0.238 \end{bmatrix} \begin{bmatrix} -0.276 \\ 1.400 \\ 1.988 \\ 1.373 \\ 0.176 \\ 1.115 \\ 1.298 \\ -0.238 \end{bmatrix} \begin{bmatrix} -0.271 \\ 1.406 \\ 2.002 \\ 1.378 \\ 0.171 \\ 1.117 \\ 1.294 \\ -0.245 \end{bmatrix} \begin{bmatrix} -0.270 \\ 1.406 \\ 2.003 \\ 1.378 \\ 0.170 \\ 1.117 \\ 1.293 \\ -0.245 \end{bmatrix} \begin{bmatrix} -0.270 \\ 1.406 \\ 2.003 \\ 1.378 \\ 0.170 \\ 1.117 \\ 1.293 \\ -0.245 \end{bmatrix} \begin{bmatrix} -0.276 \\ 1.400 \\ 1.988 \\ 1.373 \\ 0.176 \\ 1.115 \\ 1.298 \\ -0.238 \end{bmatrix} \begin{bmatrix} -0.276 \\ 1.400 \\ 1.988 \\ 1.373 \\ 0.176 \\ 1.115 \\ 1.298 \\ -0.238 \end{bmatrix} \begin{bmatrix} -0.276 \\ 1.406 \\ 1.987 \\ 1.375 \\ 0.172 \\ 1.113 \\ 1.296 \\ -0.238 \end{bmatrix}$$

**Caption:** estimated $\hat{\boldsymbol{x}}$ vectors corresponding to the algorithms presented in Table (2). Each column represents the parameter estimates derived from optimizing the logistic loss function, employing different initialization strategies and convergence criteria.

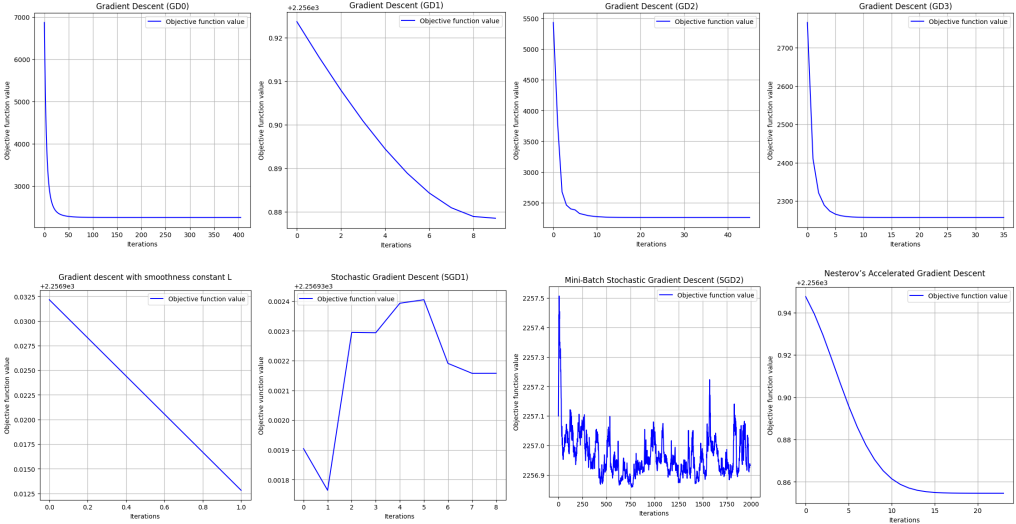


**Figure 5:** Objective function values across iterations of the gradient descent algorithm. Each image represents the evolution of the objective function at various iterations and how the algorithm converges towards the optimal solution over time.

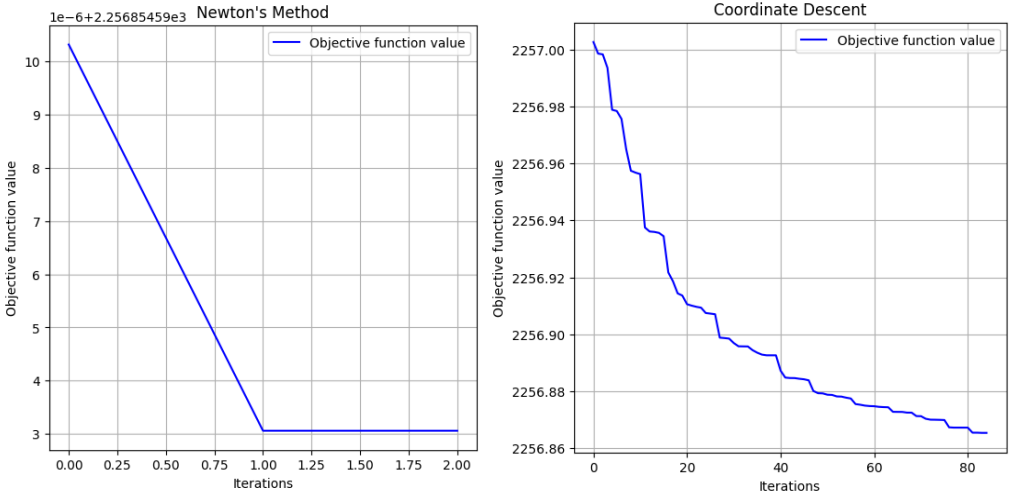


**Figure 6:** Objective function values across iterations of the gradient descent algorithm. Each image represents the evolution of the objective function at various iterations and how the algorithm converges towards the optimal solution over time.

# Conclusion

The results highlight several key insights:

- **Gradient Descent (GD)** showed consistent performance, with variations in steps and time depending on the specific variant used. In particular, **Gradient Descent Smooth** converged very quickly, requiring only 2 steps with minimal execution time, making it one of the most efficient algorithms in this context.
- **Stochastic Gradient Descent (SGD)** had more variability in performance, particularly in SGD1 where the algorithm's behavior was more influenced by noise, resulting in non-robust solutions and significant fluctuations in the number of steps and accuracy across different runs.
- **Accelerated Gradient Descent** showed strong performance, combining relatively fast convergence with lower computational cost, making it a robust option for large-scale problems.
- **Newton's Method** converged in only 2 steps but at the cost of higher computational time due to the complexity of computing second-order derivatives.
- **Coordinate Descent** was slower in terms of steps but had a reasonable execution time, providing a balance between computational efficiency and convergence rate.

Overall, the choice of optimization algorithm depends heavily on the specific problem at hand and the trade-offs between convergence speed and computational cost. For classification tasks with a smooth objective function, **Accelerated Gradient Descent** and **Newton's Method** emerge as highly effective approaches. However, for scenarios with larger datasets or noisy data, **Stochastic Gradient Descent** might require further tuning to improve robustness.

# References

[1] M. Mazzoleni, “Identificazione dei modelli e analisi dei dati (imad),” 2024.

[2] F. Maggioni, “Optimization,” 2024.

[3] T. Marco, “Logistic regression - maximum likelihood estimation,” 2021.