Optimization Methods for Data Science
Final Project Report

Asia Montico (1966494) Jacopo Caldana (2212909)

30/04/2025

1 General Introduction and Data Overview

This project explores the application of numerical optimization techniques to fundamental problems in computer vision: regression and classification. The primary objective is to implement machine learning models—specifically Multi-Layer Perceptrons (MLP) and Support Vector Machines (SVM)—from scratch, leveraging `scipy.optimize` and `cvxopt` solvers while avoiding automatic differentiation libraries. This approach allows for a deeper understanding of the underlying mathematical optimization landscape.

1.1 Dataset Overview

The empirical analysis is based on the **UTKFace** dataset, a large-scale collection of over 20,000 face images annotated with age, gender, and ethnicity. The images are collected "in the wild", providing a realistic distribution with significant variations in pose, illumination, and occlusion. To ensure robust input representation, we utilize high-dimensional feature vectors ($d = 2048$) extracted via a **ResNet convolutional backbone**, rather than processing raw RGB pixels. The data is provided in three specific subsets: `AGE_REGRESSION.csv` (continuous target), `GENDER_CLASSIFICATION.csv` (binary target), and `ETHNICITY_CLASSIFICATION.csv` (multiclass target).

2 Part 1: Multi-Layer Perceptron (Age Regression)

2.1 1. Data Analysis and Preprocessing

The first task involves predicting the biological age of a subject. We utilized the `AGE_REGRESSION.csv` dataset, which contains feature vectors of dimension $d = 2048$ (typical for ResNet-50 avg-pool layers) and a floating-point target.

Data Audit and Cleaning A preliminary audit of the dataset revealed the presence of 169 duplicate entries among the 20,475 total samples. To ensure the integrity of the evaluation and prevent data leakage between training and testing splits, these duplicates were removed. The final dataset consists of 20,306 unique samples. We verified the absence of NaN or Infinite values, ensuring numerical validity. The data was partitioned into a Training Set (80%, approx. 16,244 samples) and a Test Set (20%, approx. 4,062 samples) using a fixed random seed for reproducibility.

Feature Scaling and Target Normalization Numerical optimization algorithms, particularly quasi-Newton methods like L-BFGS, are sensitive to the scaling of input variables. Ill-conditioned Hessian matrices can lead to slow convergence. Therefore, we applied:

- **Z-Score Standardization:** Input features X were centered and scaled to unit variance based on statistics computed solely on the training set: $X_{scaled} = (X - \mu_{train})/\sigma_{train}$.
- **Min-Max Normalization on Target:** The target variable y (Age) spans a wide range $[0, 116]$. To align the target with the active region of saturation-based activation functions (such as Hyperbolic Tangent), we normalized y to the range $[0, 1]$. This prevents the "vanishing gradient" problem in the output layer during the initial phases of training.

2.2 2. Model Architecture and Mathematical Formulation

We implemented a fully connected Multi-Layer Perceptron (MLP) for regression. The network consists of an input layer, hidden layers, and a single linear output unit.

Loss Function The objective is to minimize the Mean Squared Error (MSE) augmented with an L2 regularization term (Ridge Regression penalty) to mitigate overfitting. The loss function $E(\boldsymbol{\omega})$ is defined as:

$$E(\boldsymbol{\omega}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i(\boldsymbol{x}_i))^2 + \lambda \sum_{l=1}^{L} ||\boldsymbol{W}^{(l)}||_F^2 \tag{1}$$

where N is the number of samples, $\boldsymbol{\omega}$ represents the flattened vector of all weights and biases, L represents the number of weight matrices (trainable layers), and λ is the regularization hyperparameter.

Manual Gradient Computation A critical requirement of this project was the exclusion of automatic differentiation frameworks (e.g., PyTorch, TensorFlow). Consequently, we derived and implemented the gradients ∇E manually via backpropagation. Forward propagation computes activations $A^{(l)} = \sigma(Z^{(l)})$, where $Z^{(l)} = A^{(l-1)}W^{(l)} + b^{(l)}$. Backward propagation computes the error terms $\delta^{(l)}$. For the regression output layer with linear activation, $\delta^{(L)} \propto (\hat{y} - y)$. For hidden layers, the error is propagated using the chain rule: $\delta^{(l)} = (\delta^{(l+1)}(W^{(l+1)})^T) \odot \sigma'(Z^{(l)})$. These gradients constitute the Jacobian passed to the solver.

Weight Initialization Proper initialization is vital for non-convex optimization. We employed **Xavier/Glorot Initialization** for Tanh activations. This ensures that the variance of activations remains stable across layers at the beginning of training, facilitating efficient gradient flow.

2.3 3. Model Selection (Hyperparameter Tuning)

The hyperparameter search space involves computationally expensive architectures (up to 4 layers and 128 neurons). Performing a full training cycle for every combination in a standard Grid Search proved to be computationally prohibitive given the CPU-based optimization constraint.

To address this, we implemented a **Multi-Stage Selection Strategy** (inspired by Successive Halving). This approach allocates computational resources dynamically:

1. **Exploration Phase (Round 1):** We sampled **50 random configurations** from the valid architecture space and trained them for a limited number of iterations (`maxiter=20`). This allowed us to quickly identify and discard models with poor convergence properties or bad initializations.
2. **Refinement Phase (Round 2):** The top performing models from Round 1 were selected and retrained with a higher iteration budget to confirm their generalization capability.
3. **Search Space:**

- **Hidden Layers:** 2, 3, and 4 layers.
- **Width (N_l):** Combinations of 32, 64, and 128 neurons (using non-decreasing combinations to avoid permutations).
- **Regularization (λ):** $\{10^{-2}, 10^{-3}, 10^{-4}\}$.
- **Activation Functions:** ReLU and Tanh.

Optimal Configuration The strategy identified the following optimal hyperparameters:

- **Architecture:** 3 Hidden Layers with sizes $[32, 32, 64]$.
- **Activation:** Tanh (Hyperbolic Tangent).
- **Regularization:** $\lambda = 0.0001$.

2.4 4. Optimization Routine and Results

The optimization problem was solved using the **L-BFGS-B** algorithm (Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Bounds) provided by `scipy.optimize`. L-BFGS-B is a quasi-Newton method that approximates the Hessian matrix of the loss function using updates from gradient evaluations. It is particularly well-suited for this problem because it is more sample-efficient than first-order methods (like SGD) for batch optimization and avoids the computational cost of calculating the full Hessian inverse ($O(d^3)$).

The solver successfully converged after **836 iterations**, satisfying the relative reduction condition of the objective function.

Analysis The model achieved a final Test MAPE of approximately **23.31%**. Given the complexity of age estimation from "in the wild" images, this represents a solid performance. Crucially, the Training MAPE (23.15%) and Test MAPE (23.31%) are extremely close. This indicates that the selected architecture, combined with L2 regularization ($\lambda = 10^{-4}$), successfully avoided overfitting. The model has learned generalizable features rather than memorizing the training noise. The significant reduction in the Regularized Error (from 0.1563 to 0.0154) demonstrates the effectiveness of the optimization trajectory.

Table 1: Performance of the best MLP (Age Regression)

Metric	**Training Set**	**Validation (CV)**	**Test Set**
Initial Reg. Error	0.1563	-	-
Final Reg. Error	0.0154	0.0210	-
Initial MAPE	57.90%	-	-
Final MAPE	23.15%	23.70%	23.31%

Table 2: Configuration of the best MLP

Hyperparameter / Setting	**Value**
Hidden Layers Count	3 Hidden Layers
Neurons per Hidden Layer (N_l)	$[32, 32, 64]$
Regularization Parameter (λ)	0.0001
Activation Function	Tanh
Optimization Solver	L-BFGS-B (Scipy)
Iterations	836
Optimization Time	353.49 s

3 Part 2: Support Vector Machines

3.1 Question 2: Dual SVM Implementation & Hyperparameter Tuning

3.2 Question 3: Most Violating Pair (MVP) Decomposition

3.3 Question 4 (Bonus): Multiclass Classification

3.4 Comparative Analysis