Optimization Methods for Data Science: Final Project Report

Valeria Avino Marta Lombardi

July 13, 2025

Contents

Part 1: Multi-Layer Perceptron

1.1 Optimization Routine

The core of our Multi-Layer Perceptron (MLP) training process lies in an optimization routine designed to minimize the L2-regularized Mean Squared Error (MSE) loss:

$$\mathcal{L}(\omega) = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2 + \lambda \sum_{l=1}^{L-1} \|\mathbf{W}^{(l)}\|_F^2 + \lambda\|\mathbf{v}\|^2$$

Parameter Representation and Management:

For efficient optimization, the individual weight matrices ($\mathbf{W}^{(l)}$), bias vectors ($\mathbf{b}^{(l)}$) and the final output weight vector ($\mathbf{v}$) are **unrolled** into a single, contiguous 1D NumPy array ω. This unified representation allows standard numerical optimizers to operate on the entire parameter space simultaneously. Helper functions (`unroll_params` and `roll_params`) are utilized to convert between the flattened vector and the structured list of matrices/vectors, preserving their original shapes.

Optimization Algorithm:

The minimization of $\mathcal{L}(\omega)$ is performed using `scipy.optimize.minimize`, specifically employing the **L-BFGS-B** algorithm.

The key parameters for this optimization routine are:

* **method : 'L-BFGS-B'**: A quasi-Newton algorithm chosen for its efficiency in minimizing non-linear functions by approximating second-order information, leading to faster convergence than first-order methods. It operates in a full-batch manner.
* **jac : 'True'**: Indicates that the objective function provides the analytical gradient (Jacobian), which is crucial for L-BFGS-B's efficiency and accuracy.
* **disp: 'False'**: Suppresses verbose optimization output for a cleaner log.
* **maxiter: 'self.max_iter'**: Kept to 500 for Grid Search to speed up tuning and higher (5000), for final model training to ensure thorough convergence.
* **ftol: '1e-5'**: (The function tolerance for convergence) A value of 10^{-5} is chosen to **balance optimization speed with model performance**. A looser tolerance (e.g., 10^{-4}) would converge faster but yield worse MAPE, while a tighter tolerance, as the default of 10^{-9}, would take significantly more iterations for minimal performance gains on MAPE. This value ensures good performance without excessive computation.
* **callback : 'callback_arg'**: Provides a custom function to monitor training progress by printing the non-regularized loss at specified intervals.

Gradient Computation (The Role of Forward and Backward Passes):

The L-BFGS-B algorithm requires explicit computation of the gradient of the objective function with respect to all parameters, i.e., $\nabla\mathcal{L}(\omega)$. This is achieved through the following steps:

Forward Pass (`forward` function): Given an input $\mathbf{X}$ and the current model parameters ($\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$, $\mathbf{v}$), the `forward` function propagates the input through the network's layers. Let $\mathbf{a}^{(0)} = \mathbf{X} \in \mathbb{R}^{D\times N}$ be the input features, where D is the number of features and N is the number of samples.

For each hidden layer l (from $l = 1$ to $L-1$): The pre-activation $\mathbf{a}^{(l)} \in \mathbb{R}^{N_l\times N}$ is computed as:

$$\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{N_l\times N_{l-1}}$ is the weight matrix for layer l and $\mathbf{b}^{(l)} \in \mathbb{R}^{N_l\times 1}$ is the bias vector for layer l.

The post-activation $\mathbf{z}^{(l)} \in \mathbb{R}^{N_l\times N}$ is then obtained by applying the non-linear activation function $g(\cdot)$ element-wise:

$$\mathbf{z}^{(l)} = g(\mathbf{a}^{(l)})$$

The network's final prediction $\hat{\mathbf{y}} \in \mathbb{R}^{1\times N}$ is a linear combination of the post-activations from the last hidden layer $\mathbf{z}^{(L-1)}$ and the output weights $\mathbf{v} \in \mathbb{R}^{N_{L-1}\times 1}$:

$$\hat{\mathbf{y}} = (\mathbf{v})^T\mathbf{z}^{(L-1)}$$

Crucially, it stores these intermediate activations and pre-activations as they are essential for the backpropagation step.

Backward Pass (`backward` function): The `backward` function implements the backpropagation algorithm. Starting from the error at the output layer (the derivative of the MSE loss with respect to predictions):

$$\delta^{(L)} = \frac{\partial E}{\partial \hat{y}} = -\frac{2}{N}(y - \hat{y})$$

It computes the gradient with respect to the output weights:

$$\frac{\partial E}{\partial \mathbf{v}} = \mathbf{z}^{(L-1)} \cdot (\delta^{(L)})^T$$

It backpropagates the error to the last hidden layer:

$$\delta^{(L-1)} = \left(\mathbf{v}\delta^{(L)}\right) \odot g'\left(\mathbf{a}^{(L-1)}\right)$$

For each hidden layer $l \in \{L-1, \ldots, 1\}$, it recursively computes:

$$\frac{\partial E}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} \cdot (\mathbf{z}^{(l-1)})^T$$

$$\frac{\partial E}{\partial \mathbf{b}^{(l)}} = \sum_{j=1}^{N} \delta_j^{(l)}$$

$$\delta^{(l-1)} = (\mathbf{W}^{(l)})^T \cdot \delta^{(l)} \odot g'\left(\mathbf{a}^{(l-1)}\right)$$

This process propagates the error signal backward through the network, layer by layer, utilizing the stored intermediate activations and the derivatives of the activation functions.

Gradient Regularization: The gradient of the L2 regularization term, $\frac{\partial}{\partial\omega}\left(\lambda\sum\|\omega\|_F^2\right) = 2\lambda\omega$, is calculated separately for each weight matrix and the final output vector. These regularization gradients are then added to the corresponding gradients derived from the MSE loss (computed by the `backward` function) to form the total gradient $\nabla\mathcal{L}(\omega)$. This combined gradient is then flattened for the optimizer.

Iterative Optimization:

During each iteration, `scipy.optimize.minimize` (using L-BFGS-B) receives the current flattened parameter vector ω. It then calls our 'objective-function' to obtain both the current loss $\mathcal{L}(\omega)$ and its total gradient $\nabla\mathcal{L}(\omega)$. Based on these values, the optimizer computes an optimal step direction and size to update ω, aiming to descend towards the minimum of the objective function. This process continues until a convergence criterion is met (e.g., gradient norm falls below a tolerance) or a maximum number of iterations (`maxiter`) is reached.

1.2 Results

1.2.1 Final Setting for Hyperparameters

The final settings for the selected non-linearity, total number of layers (L), neurons per layer (N_l) and regularization factor (λ) are described in 1.3.
These hyperparameters were chosen using a **5-fold Cross-Validation Grid Search** approach. The search space for this process encompassed:

- Total Number of Layers (L): $[3, 4, 5]$ (including output layer)
- Neurons per Hidden Layer (N_l):
  For $L = 3$: $[32, 16], [32, 32], [64, 32]$
  For $L = 4$: $[16, 16, 16], [32, 16, 32], [32, 32, 32]$
  For $L = 5$: $[16, 8, 16, 8], [32, 16, 32, 16]$
- Activation Function: `[tanh (g1), sigmoid (g2)]`
- Regularization Factor (λ): $[0.001, 0.01]$

The optimal configuration was identified as the combination of hyperparameters that yielded the **lowest average Mean Absolute Percentage Error (MAPE)** across the validation folds.

$$MAPE = \frac{100}{N}\sum_{i=1}^{N}\left|\frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}}\right|$$

To enhance computational efficiency during the grid search, the `max_iter` parameter for the L-BFGS-B optimizer was strategically set to **500 iterations**. This allowed for a faster ranking of candidate configurations. Subsequently, the chosen **optimal model** was trained on the entire training dataset with a higher `max_iter` of **5000 iterations** to ensure full convergence and robust performance (however converged in just 337!).

1.2.2 Performance and Configuration Tables

Table 1.1: Performance of the best MLP

PERFORMANCE					
FINAL TRAIN LOSS	FINAL TEST LOSS	FINAL TRAIN MAPE	FINAL TEST MAPE	AVG VAL MAPE	OPTIMIZATION TIME
93.0976	94.7284	20.1776%	20.0649%	20.3638%	18.29 s

Note: Final train and test loss here refer to the non-regularized MSE loss (no L2 regularization term). The regularized losses can be found in the notebook.

Table 1.2: Configuration of the best MLP

SETTINGS						
HIDDEN LAYER 1		HIDDEN LAYER 2		HIDDEN LAYER 3		OTHER
Number of Neurons	Nonlinearity	Number of Neurons	Nonlinearity	Number of Neurons	Nonlinearity	Regularization Term
16	sigmoid	16	sigmoid	16	sigmoid	0.001

Best-performing architecture during validation: a symmetric structure with **three hidden layers**, each containing **16 neurons** and using the **sigmoid activation** function

Table 1.3: Optimization Details

Optimization parameter	**Chosen Value**
Optimization Solver	L-BFGS-B
Max Iterations	5000
Number of Iterations	337
Optimization Time	18.29 s
Starting Objective Value	1742.7
Final Objective Value	94.165
Initialization	Xavier/Glorot initialization (scaled random normal)
Function tolerance for convergence	10^{-5}
Optimization Message	*CONVERGENCE: REL_REDUCTION_OF_F≤*EPSMCH*

Note: The optimizer terminated successfully after 337 iterations with a convergence criterion based on minimal relative reduction in the objective function. The optimization message indicates that the relative improvement in the objective function became smaller than the predefined numerical threshold (`ftol` = 10^{-5}), and the algorithm considered the solution to be sufficiently optimal.

Part 2: Support Vector Machine (SVM)

1.3 Question 2 - Binary Classification via Dual SVM

1.3.1 Problem Setup and Optimization

We consider a binary classification task using the `GENDER_CLASSIFICATION.csv` dataset, where the original labels $\{0, 1\}$ are mapped to $\{-1, +1\}$. The implementation is based on the `SVM` class provided in the file `Functions_22_Avino_Lombardi.py`. The dual formulation of the Support Vector Machine problem is given by:

$$-\min_{\substack{\boldsymbol{\alpha}\in\mathbb{R}^n \\ 0\le\alpha_i\le C\ \forall i \\ \sum_{i=1}^n \alpha_i y_i = 0}} \left\{ \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n}\alpha_i \right\}$$

where $K(\cdot,\cdot)$ denotes a generic positive semi-definite kernel and $C > 0$ is a regularization parameter. The bias term b is computed as the average of the optimality conditions on support vectors strictly lying on the margin. Specifically, for indices k such that $0 < \alpha_k < C$, we use:

$$b = \frac{1}{|\mathcal{SV}|}\sum_{k\in\mathcal{SV}} \left(y_k - \sum_{i=1}^{n}\alpha_i y_i K(x_i, x_k) \right), \quad \mathcal{SV} := \{k : 0 < \alpha_k < C\}$$

The resulting decision function is:

$$\hat{y} = \text{sign}(f(x) + b), \quad \text{where} \quad f(x) = \left(\sum_{i\in\mathcal{S}} \alpha_i y_i K(x_i, x) \right) \quad \text{and} \quad \mathcal{S} := \{i : \alpha_i > 0\}$$

1.3.2 Training and Hyperparameter Selection

Model training is carried out using the `cvxopt.solvers.qp` routine from the `CVXOPT` library. All solver parameters are set to **DEFAULT**, except for disabling the solver output (`show_progress = False`). We also provide an initial guess of zero for the Lagrange multipliers via the `initvals` argument, which implies that the initial value of the dual objective function is always zero.
We consider the following kernel:

- **Polynomial kernel:** $K(x_i, x_j) = (x_i^\top x_j + 1)^p$
- **Gaussian RBF kernel:** $K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$

We evaluate different kernel configurations via 5-fold cross-validation on the training set:

$$C \in [1, 10], \quad p \in \{2, 3, 4\}, \quad \gamma \in \{0.1, 0.5, 1\}$$

The best-performing configuration for the polynomial kernel is $C = 1$ and $p = 2$ (validation accuracy: 0.9162), while for the RBF kernel it is $C = 1$ and $\gamma = 0.1$ (validation accuracy: 0.9212). We observe that high values of C and p or γ tend to cause overfitting, leading to low validation performance despite high training accuracy. However, very low values may cause underfitting. Thus, a proper trade-off between model complexity and generalization ability is essential.

1.4 Question 3 – Training via MVP-SMO Algorithm

We replace the previously used interior-point solver with an iterative approach based on the **Sequential Minimal Optimization (SMO)** framework. Our implementation closely follows the method proposed by Platt [1] and is described in detail in the following sections. Specifically, our SMO algorithm, available in the file `Functions_23_Avino_Lombardi.py` as the class `MVP`, employs the **Most Violating Pair (MVP)** strategy to iteratively select the pair of dual variables whose update leads to the greatest reduction in the violation of the **Karush-Kuhn-Tucker (KKT)** conditions. The optimization starts with $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{b} = \mathbf{0}$, **gradient** $= -\mathbf{1}$ for all points and an **initial objective** value of $\mathbf{0}$. We use the hyperparameters selected via cross-validation in Question 2 for the best-performing kernel, namely the Gaussian kernel with $C = 1$ and $\gamma = 0.1$.

1.4.1 Pair Selection via KKT Violation

Let us define the dual residuals

$$G_k := 1 - y_k f(x_k) = 1 - y_k \left(\sum_{l=1}^{n} \alpha_l y_l K(x_l, x_k) + b \right)$$

and the index sets

$$R = \{k \mid (\alpha_k = 0 \wedge y_k = 1) \vee (\alpha_k = C \wedge y_k = -1) \vee (0 < \alpha_k < C)\},$$
$$S = \{k \mid (\alpha_k = 0 \wedge y_k = -1) \vee (\alpha_k = C \wedge y_k = 1) \vee (0 < \alpha_k < C)\}.$$

We compute

$$m := \max_{k \in R} \frac{G_k}{-y_k}, \quad M := \min_{k \in S} \frac{G_k}{-y_k}$$

and identify the pair $(i, j) \in R \times S$ such that $\dfrac{G_i}{-y_i} = m$ and $\dfrac{G_j}{-y_j} = M$. This pair corresponds to the maximal violation of the KKT complementary slackness conditions. Convergence is declared when the violation gap $m - M$ falls below a fixed tolerance (default $\varepsilon = 10^{-3}$), or if no valid pair in $R \times S$ can be found. Additionally, early stopping is triggered in the following cases: numerically unstable curvature ($\eta \leq 10^{-9}$), negligible update magnitude ($|\Delta\alpha_j| < 10^{-5}$), or absence of eligible pairs.

1.4.2 Pairwise Update under Box Constraints

Given the selected pair (i, j), we define:

$$E_k := f(x_k) - y_k, \quad \eta := K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j), \quad a := y_j(E_i - E_j).$$

To preserve the constraint $\sum_i \alpha_i y_i = 0$, we set:

$$\alpha_i^{\text{new}} = \alpha_i^{\text{old}} + y_i y_j (\alpha_j^{\text{old}} - \alpha_j^{\text{new}}).$$

The updated α_j is obtained by projecting the unconstrained solution onto the feasible interval

$$\alpha_j^{\text{new}} = \min \left(\max \left(\alpha_j^{\text{old}} - \frac{a}{\eta}, L \right), H \right),$$

with bounds $[L, H]$ determined by label configuration

$$y_i \neq y_j: \quad L = \max(0, \alpha_j^{\text{old}} - \alpha_i^{\text{old}}), \quad H = \min(C, C + \alpha_j^{\text{old}} - \alpha_i^{\text{old}}),$$
$$y_i = y_j: \quad L = \max(0, \alpha_j^{\text{old}} + \alpha_i^{\text{old}} - C), \quad H = \min(C, \alpha_j^{\text{old}} + \alpha_i^{\text{old}}).$$

1.4.3 Incremental Update of Residuals and Bias

To avoid recomputing $f(x_k)$, we incrementally update the residuals:

$$G_k \leftarrow G_k + \Delta\alpha_i Q_{ki} + \Delta\alpha_j Q_{kj}, \quad Q_{kl} = y_k y_l K(x_k, x_l),$$

where $\Delta\alpha_l = \alpha_l^{\text{new}} - \alpha_l^{\text{old}}$. The bias is corrected via:

$$b = \begin{cases} b_1 = b - E_i - y_i\Delta\alpha_i K_{ii} - y_j\Delta\alpha_j K_{ij} & \text{if } 0 < \alpha_i^{\text{new}} < C, \\ b_2 = b - E_j - y_i\Delta\alpha_i K_{ij} - y_j\Delta\alpha_j K_{jj} & \text{if } 0 < \alpha_j^{\text{new}} < C, \\ \dfrac{b_1 + b_2}{2} & \text{otherwise.} \end{cases}$$

1.5 Question 4 - Multiclass Classification Strategy

To address the multiclass setting, we implemented a dedicated class named `MulticlassSVM`, located in the file `Functions_24_Avino_Lombardi.py`. This class generalizes the binary MVP solver developed in Question 3 to multiclass problems, following two classical decomposition strategies: **One-vs-Rest (OvR)** and **One-vs-One (OvO)**. The experiments were conducted on a subset of the `ETHNICITY.csv` dataset, restricted to three out of the five available ethnic groups to define a three-class classification problem. Let $K = 3$ denote the number of classes.
In the **OvR** setting, `MulticlassSVM` trains K binary classifiers. For each class $c_k \in \{1, \ldots, K\}$, the binary labels are defined as:

$$y_i^{(k)} = \begin{cases} +1 & \text{if } y_i = c_k \\ -1 & \text{otherwise} \end{cases}$$

Each classifier is trained independently using the MVP algorithm, which optimizes the dual problem through the **Sequential Minimal Optimization** scheme. The final difference $m(\lambda) - M(\lambda)$, averaged across the trained binary models, is 0.361765. At inference time, the class label $\hat{y}$ for a new input x is determined by selecting the classifier with the highest decision score:

$$\hat{y} = \arg\max_k \left(\sum_{i=1}^{n_k} \alpha_i^{(k)} y_i^{(k)} K(x_i, x) + b^{(k)} \right)$$

In the **OvO** strategy, the class trains one binary classifier for each unordered pair (c_i, c_j), for a total of $\binom{K}{2} = 3$ models. Each classifier is trained only on the subset of samples belonging to classes c_i and c_j, and assigns labels $+1$ and -1 respectively. At prediction time, each classifier casts one vote, and the class receiving the majority of votes is returned:

$$\hat{y} = \arg\max_c \#\text{votes for class } c$$

Internally, the class stores a dictionary of trained MVP instances: one per class (OvR) or one per pair (OvO). The function `_get_mvp_decision_function_scores` is used to reconstruct the raw decision function from each MVP model:

$$f(x) = \sum_{i=1}^{n_{SV}} \alpha_i y_i K(x_i, x) + b$$

This enables prediction without modifying the MVP class itself. Due to the specific structure of the dataset and the shift from binary to multiclass classification, the hyperparameters selected in Question 2 were not directly transferable. We therefore performed a new 5-fold cross-validation procedure to jointly select the best values for C, the kernel type (Gaussian or polynomial), its associated parameters (γ or p) and the classification strategy (OvR or OvO), using validation accuracy as the selection criterion. The final model was then retrained on the full training set using the optimal configuration and evaluated on a held-out test set.

1.6 Results

Table 1.4: Configuration and result of the best SVM

QUESTION	HYPERPARAMETERS			ML PERFORMANCE		OPTIMIZATION PERFORMANCE		
	KERNEL	C	p or γ	TRAIN ACCURACY	TEST ACCURACY	FINAL OBJECTIVE	NUMBER OF ITERATIONS	CPU TIME
Q2	polynomial	1	2	0.92	0.905	-126.227	15	1.1888
Q2	gaussian	1	0.1	0.9212	0.91	-135.4057	14	0.9883
Q3	gaussian	1	0.1	0.92	0.91	-134.3861	102	0.1024
Q4	gaussian	1	0.1	0.9167	0.9067	-163.921259	932	1.3609

Bibliography

[1] J. Platt. *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. In *Advances in Kernel Methods—Support Vector Learning*, volume 208, 1998.