Report on Summative Exercise On Linear Classification and MLP Regression

Drago Jakimovski

April 30, 2025

1 Linear Classification Using Regularized SVM

1.1 Introduction

In this task, we aim to build a linear classifier to predict loan approval decisions (approved = 1, rejected = 0) using a dataset that includes 45,000 instances and 14 features related to applicant demographics, credit history, and loan details. Our goal is to accurately classify the binary outcome variable `loan_status` based on these input features.

1.1.1 Objective Function: Regularized SVM

We use a linear Support Vector Machine (SVM) with L2 regularization to formulate the classification problem. The objective function we minimize is a combination of the hinge loss and a regularization term:

$$\mathcal{O} = C \sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i + w_0)) + \frac{1}{2}\mathbf{w}^T \mathbf{w}$$

Here, $\mathbf{w}$ is the weight vector, w_0 is the bias term, $y_i \in \{-1, 1\}$ is the true label, $\mathbf{x}_i$ is the input vector, and C is the regularization strength. The first term penalizes misclassified or margin-violating samples using hinge loss, while the second term discourages large weights to prevent overfitting. The parameter C balances the two objectives.

The linear prediction model is defined as:

$$\hat{y} = w^T \mathbf{x} + w_0$$

1.1.2 Gradient Computation

To minimize the objective function, we compute its gradient. For a given data point, the hinge loss contributes to the gradient only when:

$$y_i(\mathbf{w}^T \mathbf{x}_i + w_0) < 1$$

In such cases, the gradient of the hinge loss with respect to $\mathbf{w}$ is:

$$\nabla_{\mathbf{w}} = -y_i \mathbf{x}_i$$

The gradient of the regularization term is:

$$\nabla_{\mathbf{w}} = \mathbf{w}$$

Thus, the total gradient used in training is the sum of the gradients from the hinge loss and the regularization term.

1.1.3 Gradient Descent and Training Implementation

We implement training using batch gradient descent in the function `linear_gd_train`. At each iteration, the weights are updated using the rule:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla \mathcal{O}$$

where η is the learning rate. The objective value is computed and stored at each step. For the gradient update rule a for loop has to be used with N_{max} iterations but for all matrix and vector multiplications numpy function were used for more optimized code.

Figure 1 confirms the proper implementation of our objective and optimization routine, as the cost consistently decreases over 200 iterations.

1.1.4 Preliminary Results and Motivation for Tuning

We divide our data set into an 80% training and 20% testing split and standardize both datasets to prepare for training and testing. Using our model with $C = 0.2$ and $\eta = 0.001$, we achieve the following:

- **Test Accuracy:** 0.4477
- **Test F1 Score:** 0.3352

When we increase the learning rate to $\eta = 0.02$, performance significantly improves:

- **Test Accuracy:** 0.8391
- **Test F1 Score:** 0.6854

While accuracy shows general performance, the F1 score is more informative in imbalanced datasets. Our dataset has a class distribution of approximately 87.5% to 12.5%, making accuracy potentially misleading (e.g., a model predicting only the majority class achieves 87.5% accuracy but 0 F1 score for the minority class). Therefore, F1 score is the preferred metric for evaluation.

These initial results highlight the sensitivity of model performance to hyperparameters C and η, motivating a systematic grid search to identify optimal values.

1.2 Learning Rate η: Effect on Training and Model Performance

To understand the impact of the learning rate η on training dynamics and model performance, we performed an initial experiment by varying η across the following values:

$$\eta \in \{0.001, 0.004, 0.008, 0.02, 0.05\}$$

As illustrated in **Figure 2**, higher learning rates lead to a faster decrease in training cost per iteration, confirming that the gradient descent optimization algorithm converges more rapidly when η is larger.

However, training speed alone does not guarantee better model performance. The classification results on the test set for each learning rate are summarised below:

- $\eta = 0.001$: Test Accuracy = 0.4410, Test F1 Score = 0.2861
- $\eta = 0.004$: Test Accuracy = 0.4364, Test F1 Score = 0.2976
- $\eta = 0.008$: Test Accuracy = 0.4477, Test F1 Score = 0.3352
- $\eta = 0.02$: Test Accuracy = 0.8391, Test F1 Score = 0.6854
- $\eta = 0.05$: Test Accuracy = 0.7778, Test F1 Score = 0.0000

From these results, we observe that:

- Very small η values result in slow convergence and underfitting, yielding poor accuracy and F1 scores.
- Increasing η to 0.02 improves both training speed and generalization performance significantly.
- However, a too-large learning rate ($\eta = 0.05$) causes the model to become unstable, possibly overfitting to the majority class and failing to detect the minority class, as seen by the F1 score dropping to 0.

Conclusion: While higher η values accelerate training, they must be chosen carefully. An optimal value like $\eta = 0.02$ balances convergence speed and generalization, especially in imbalanced classification tasks where the F1 score is a more reliable metric than accuracy.

1.3. Grid Search: Combined Effect of Learning Rate η and Regularization Strength C (additional experiment)

Building on our previous findings, it is clear that hyperparameters play a crucial role in shaping both the training dynamics and the final performance of our linear model. To further investigate, we conducted an additional experiment

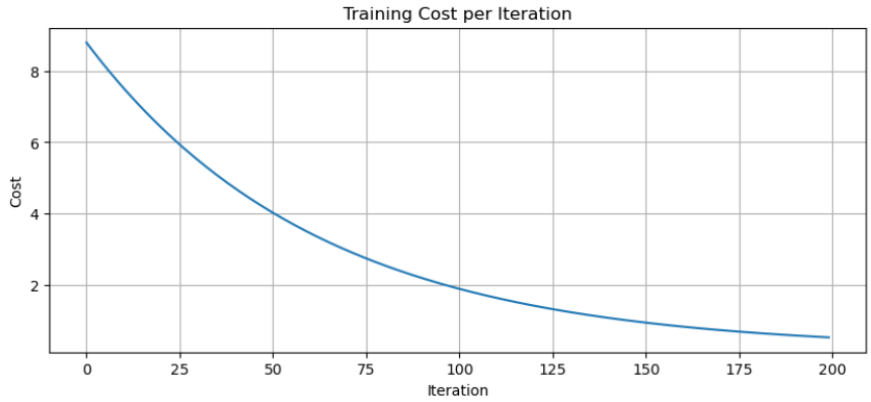

Figure 1: Training cost per iteration, showing the convergence of our gradient descent algorithm.

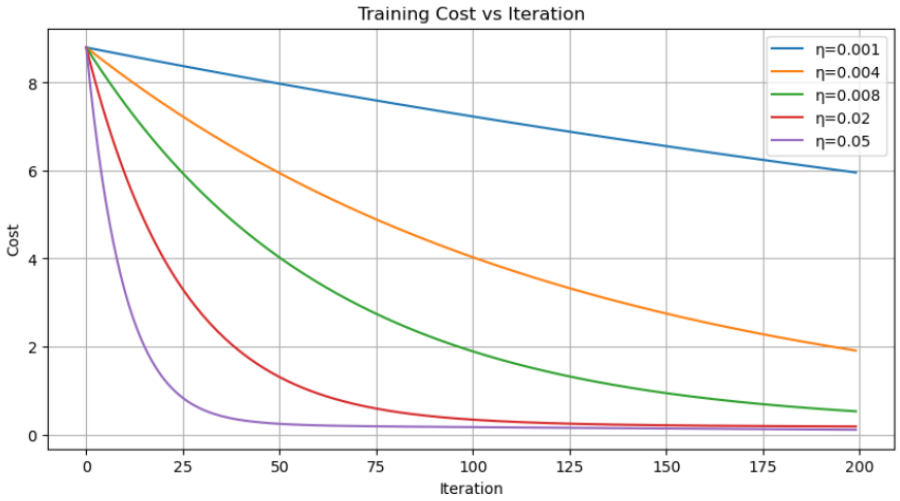

Figure 2: Training Cost vs Iteration for varying learning rates η

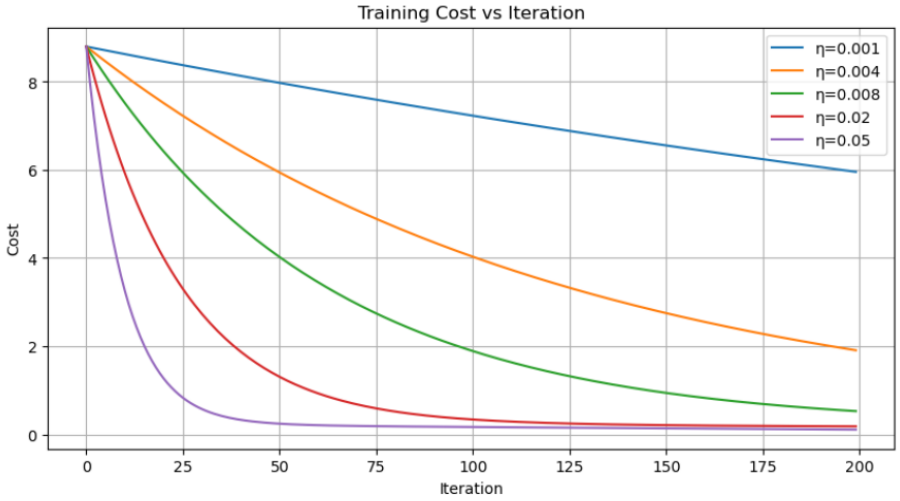

Figure 3: Training Cost vs Iteration for varying learning rates η

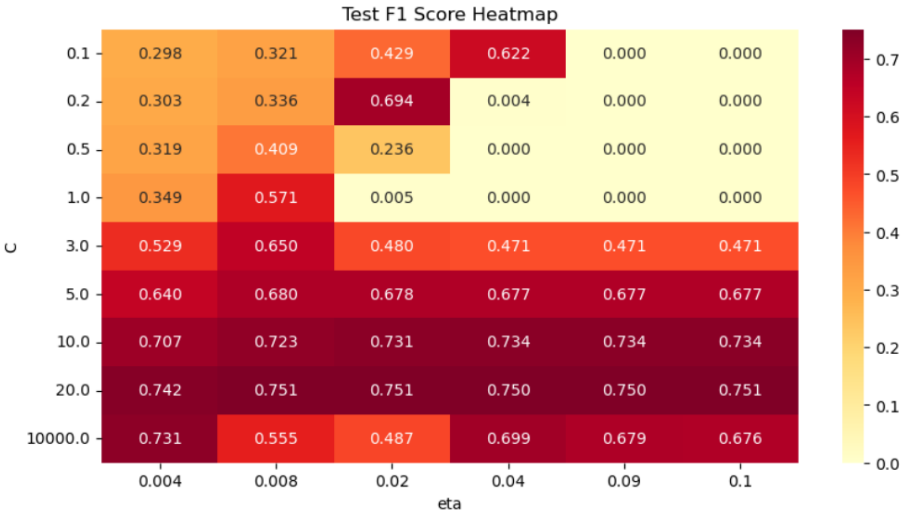

Figure 4: Test F1 score heatmap across combinations of regularization strength C and learning rate η.

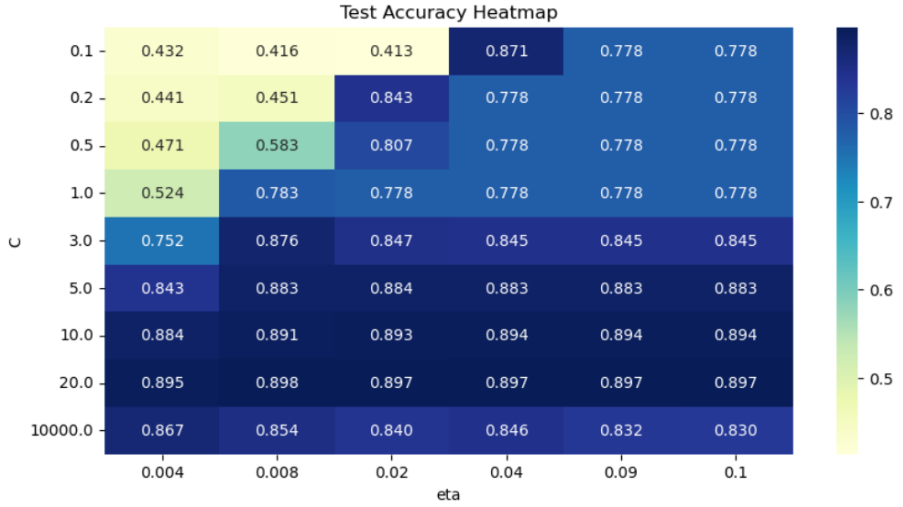

Figure 5: Accuracy heatmap across combinations of regularization strength C and learning rate η.

that systematically explores the combined effects of the learning rate η and the regularization strength C on model behavior.

In this experiment, we performed a grid search over the following values:

$$\eta \in \{0.004, 0.008, 0.02, 0.04, 0.09, 0.1\}, \quad C \in \{0.1, 0.2, 0.5, 1, 3, 5, 10, 20, 10000\}$$

The regularization strength C controls the balance between maximizing the margin and minimizing classification errors. Smaller values of C emphasize stronger regularization, leading to wider margins that tolerate more misclassifications, which can promote better generalization. Conversely, larger values of C shift the focus toward minimizing training error by reducing misclassifications but will can shrink the margin and increase the risk of overfitting. When C becomes excessively large, regularization is effectively ignored, and performance may degrade due to the model overly tailoring to the training data.

From the results visualized in Figures 4 and 5, we observe that our expectations are largely confirmed. For low values of C (e.g., $C = 0.1$), test accuracy and F1 scores remain relatively low across all learning rates due to underfitting. As C increases, both accuracy and F1 scores improve significantly, reaching optimal performance for intermediate values (e.g., $C = 10$ or $C = 20$) and stable learning rates such as $\eta = 0.02$. Beyond this range, excessively large C values begin to show signs of instability or overfitting, particularly at higher learning rates.

Conclusion: This experiment highlights the importance of jointly tuning C and η. An optimal combination is essential to achieve a good balance between bias and variance, especially in imbalanced classification settings. While

increasing C generally improves F1 score by reducing false negatives, it must be done in tandem with an appropriate learning rate to ensure convergence and generalization.

2 Soybean Production Prediction by MLP

2.1 Model Selection Analysis – Accuracy, Efficiency, and Complexity

Before we continue with the analysis, a brief note on neural networks: Due to their use of non-linear activation functions—which allow them to learn complex, non-linear patterns—and their large number of tunable training parameters (weights and biases), neural networks are capable of approximating virtually any continuous function. This is supported by the Universal Approximation Theorem, which states that a feedforward neural network with at least one hidden layer and a finite number of neurons can approximate any continuous function on a compact domain, given suitable parameters.

In this experiment, we leverage this theoretical foundation along with existing optimization libraries that implement efficient versions of gradient descent and other algorithms to help us find the best-performing neural network architecture.

In Section 2.2, we performed hyperparameter tuning for an `MLPRegressor` using a grid search across 24 configurations, varying in architecture, activation function, and number of training iterations. The best-performing model had a single hidden layer with 100 neurons, ReLU activation, and 50 training iterations. This configuration achieved a **mean squared error (MSE) of 0.0009** and an **R^2 score of 1.0000** on the test set, indicating nearly perfect prediction performance.

From a training efficiency perspective, this model is lightweight and fast to train. Despite using only 50 iterations, it converged successfully due to the structured nature of the dataset. Compared to deeper or more iterative configurations, it achieved high accuracy at a low computational cost.

In terms of model complexity, the selected architecture (100 neurons in one hidden layer) balances representational power with generalization ability. More complex models, such as two-layer MLPs, did not yield performance improvements, likely due to redundancy or overfitting. Overall, the selected model offers **high performance with minimal complexity and training cost**.

2.2 Feature Selection and Validation – Design Process and Findings

In Section 2.3, we used a `GradientBoostingRegressor` to calculate feature importance scores. This method quantified how much each input feature contributed to the prediction of soybean yield. The top five features identified

were: **Biological Weight (BW)**, **Weight of 300 Seeds (W3S)**, **ChlorophyllA663**, **Number of Pods (NP)**, and **Sugars (Su)**.

These five features were then used to train a new MLPRegressor, using the same hyperparameters as the optimal model from Section 2.2. After standardizing both the input and output data, the reduced model achieved a **test MSE of 1408.6150** and an **R^2 score of 0.9991**. Although the MSE increased slightly, the R^2 score indicates that predictive performance remained very strong.

This experiment demonstrates that a well-chosen subset of features can retain high accuracy while reducing model complexity and input dimensionality. Such feature reduction is particularly useful in real-world applications where lower data collection costs and improved model interpretability are desired. The use of permutation importance and bar plots provided visual evidence supporting the relevance of the selected features.

2.3Handling Missing and Noisy Features

Although no missing values were present in the dataset (as verified using `describe()`), several preprocessing steps were applied to ensure robustness. The `Parameters` column, which encoded categorical information, was parsed into individual components (genotype, salicylic acid level, water stress) and one-hot encoded to ensure numerical compatibility with machine learning models.

All numerical features were standardized using `StandardScaler` to prevent scale-based bias during MLP training. To address potential noise from irrelevant features, we applied feature selection using Gradient Boosting scores, which effectively filtered out the least informative inputs. This helped enhance generalization and reduce overfitting in the reduced-feature experiment.