Kernelized Linear Classification

Machine Learning and Statistical Learning

Edoardo Zanone, 33927A

Data Science for Economics, Universita' degli Studi di Milano

edoardo.zanone@studenti.unimi.it

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study

1 Introduction

This project aims to explore and implement fundamental machine learning classification algorithms from scratch, focusing on numerical feature-based label classification. The evaluation metric for the models is the 0-1 loss, ensuring consistency in assessing their performance.

The key objectives include:

- Understanding and preprocessing the dataset while avoiding data leakage between training and test sets.
- Implementing core classification algorithms without relying on libraries such as Scikit-learn:
  - **The Perceptron**
  - **Support Vector Machines (SVMs) using the Pegasos algorithm**
  - **Regularized logistic classification** (Pegasos with logistic loss instead of hinge loss)
- Testing and evaluating these models' performance.
- Enhancing performance using polynomial feature expansion (degree 2) and analyzing the resulting linear weights.
- Extending the implementation to **kernel methods**, specifically:
  - **Kernelized Perceptron** with Gaussian and polynomial kernels
  - **Kernelized Pegasos** for SVM with Gaussian and polynomial kernels

2 Data Preprocessing

To ensure the dataset is well-prepared for training and evaluation, multiple preprocessing steps were performed. These steps help maintain data quality, prevent inconsistencies, and ensure a fair comparison between models.

2.1 Loading and Inspecting the Dataset

The dataset was initially loaded and then split into features (X) and labels (Y). It consists of ten thousands data points, with binary labels that are approximately equally distributed between the values +1 and -1. A check for missing values was performed, revealing that no missing values were present, thus eliminating the need for imputation.

2.2 Outlier Detection and Removal

Outliers can significantly affect the performance of machine learning models. In order to improve the quality of our dataset and the reliability of our models the IQR (Interquartile Range) Method was applied. This technique identifies and removes instances with feature values that deviate significantly from the normal distribution of the dataset. This ensures that extreme values do not disproportionately influence model training. The IQR is defined as the difference:

$$\text{IQR} = Q_3 - Q_1$$

where:

- Q_1 is the first quartile (25th percentile),
- Q_3 is the third quartile (75th percentile).

To identify potential outliers, the lower and the upper bounds were defined as:

$$\text{Lower Bound} = Q_1 - 1.5 \times \text{IQR}$$

$$\text{Upper Bound} = Q_3 + 1.5 \times \text{IQR}$$

Approximately five hundred (5% of the dataset) data points were excluded. After the removal, the labels are 48% with value +1 and 52% with value -1

2.3 Train-Test Splitting

The dataset was split into training and test sets using a stratified approach, where 70% of the data was allocated for training and the remaining 30% for testing.

2.4 Feature Standardization

To ensure that all numerical features have similar scales, the dataset was standardized. The standardization process rescales all features to have zero mean and variance equal to one, improving the stability and performance of the learning algorithms. A crucial aspect of this approach is that the test set is standardized using the mean and standard deviation computed from the training set. Given a training set X_{train} and a test set X_{test}, these are the steps to follow:

- Compute the mean and standard deviation of the j-th feature in the training set:

$$\mu_{\text{train},j} = \frac{1}{N} \sum_{i=1}^{N} X_{\text{train},i,j}$$

$$\sigma_{\text{train},j} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (X_{\text{train},i,j} - \mu_{\text{train},j})^2}$$

  where $\mu_{\text{train},j}$ and $\sigma_{\text{train},j}$ are the mean and standard deviation of the j-th feature in the training set, and N is the number of training samples.

- Standardize the j-th feature in both the training and test sets:

$$X'_{\text{train},i,j} = \frac{X_{\text{train},i,j} - \mu_{\text{train},j}}{\sigma_{\text{train},j}}$$

$$X'_{\text{test},i,j} = \frac{X_{\text{test},i,j} - \mu_{\text{train},j}}{\sigma_{\text{train},j}}$$

This prevents data leakage, ensuring that no information from the test set influences the training process.

2.5 Handling Highly Correlated Features

Strong correlations between features can introduce redundancy and negatively impact model performance. To address this, the correlation matrix was computed for the training set, and all the features with a coefficient of 0.9 or higher (or below -0.9) were identified as highly correlated and taken into consideration. Among the ten variables only three had a coefficient large (or small) enough to be considered for analysis.

Variable 1	**Variable 2**	**Correlation Coefficient**
3	6	-0.989999
3	10	-0.979744
6	10	0.989808

Table 1: Highly Correlated Variables and their Correlation Coefficients

To determine which variables needed to be removed, it was first necessary to develop the models and apply them to the dataset, considering all possible variations resulting from the elimination of one or more features. Additionally, parameter tuning (which will be explained in the following paragraphs) was performed to optimize model performance. Once this became possible, the 0-1 losses from the various evaluations were sorted from the smallest to the largest. The outcome showed that the best combination of removed variables, leading to the lowest loss, was achieved when the sixth and tenth variables were excluded. Based on this analysis, these two variables were removed from the study. Two of these three models showed an improvement in performance after removing these features.To ensure consistency, the same dataset (without the sixth and tenth variables) was used across all models, even though the Logistic Classification and, above all, Pegasos would have performed better with different feature removal choices. Table 2 reports the 0-1 Loss before feature removal and after feature removal.

Model	**0-1 Loss (Before Removal)**	**0-1 Loss (After Removal)**
Logistic Regression	0.6792	0.4776
Pegasos SVM	0.3021	0.3169
Perceptron	0.3489	0.3193

Table 2: Comparison of 0-1 Loss Before and After Removing Variables 6 and 10

3 Methodology

To fully understand the meaning and the results of this study it is first needed to know and understand the underlying algorithms.

3.1 Perceptron Algorithm

The Perceptron algorithm is a supervised learning algorithm used for binary classification tasks. It aims to find a linear decision boundary that separates the two classes. Given a training sample (x_i, y_i), the algorithm updates the weights during training every time a misclassification occurs (if $\hat{y} \neq y$), using the following update rule:

$$w = w + \eta \; y_i \; x_i$$

where:

- w represents the weights,
- η is the learning rate,
- y_i is the true label,
- x_i is the input feature .

This process continues iteratively until the model converges, adjusting the weights to minimize the classification error.

3.2 Support Vector Machines (SVM) using Pegasos algorithm

Support Vector Machine (SVM) is a supervised classification model. The goal of the SVM is to find a hyperplane that maximizes the margin between two classes while minimizing classification errors. The objective function can be expressed as:

$$\min_{w} \quad \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i \, w^\top x_i\right),$$

where :

- w is the weight vector to be optimized,
- n is the number of training examples,
- λ is the regularization parameter,
- y_i is the true label,
- x_i is the input feature.

The Pegasos performs stochastic gradient algorithm on the objective function. It initializes $w_1 = 0$, then on the i-th iteration it chooses a random example (x_i, y_i) and then update the weight based on the sub-gradient.

3.3 Regularized Logistic Classification

Regularized Logistic Classification is a supervised learning algorithm for binary classification tasks. In this approach, regularization is applied in order to prevent overfitting by penalizing large weights. In this study it was implemented in a manner similar to the Pegasos algorithm, but with a key difference: instead of using the hinge loss funztion, the logistic function was used. The logistic loss is defined as follows:

$$\ell(y, \hat{y}) = \log_2\left(1 + e^{-y\hat{y}}\right)$$

This function measures the error between the predicted value $\hat{y}$ and the true label y. Therefore, the objective function is the same as previously described, but with the hinge loss function replaced by the logistic loss.

3.4 Polynomial Feature Expansion

This expansion is applied with the main goal of capturing non-linear relationships in the data. This technique, by transforming the original feature set, aims to reveal patterns and interactions between variables that aren't apparent in the linear form. Limited to degree 2, it transforms the original features generating higher-degree polynomial terms. As a result, the new feature set not only includes the 8 original variables, but also their squared values and their pairwise interaction. At the end of the expansion the new dataset will have 44 variables. Since the highly correlated variables have already been removed, verifying their correlation is no longer a concern.

3.5 Kernel Methods

Kernel methods are techniques that allow us to learn nonlinear relationships by implicitly mapping the input data into a higher-dimensional space. This transformation is achieved without explicitly computing the new feature representation, thanks to the kernel trick. Two widely used kernels in this project are the **Gaussian kernel** and the **polynomial kernel**.

The **Gaussian kernel** is defined as:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

where σ is a positive hyperparameter that controls the spread of the kernel.

The **polynomial kernel** is given by:

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + 1)^n, \forall n \in \mathbb{N}$$

where n is the polynomial degree .

3.5.1 Kernelized Perceptron

When applying the Perceptron algorithm with kernel methods it can be extended to handle nonlinear classification tasks. Instead of learning explicit weights for each feature, the Kernelized Perceptron maintains a history of misclassified examples and computes predictions using their kernelized similarity to new inputs. Given a training set, the decision function is expressed as:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

where α_i are the weights and y_i are their corresponding labels. This allows the Perceptron to classify data in a transformed feature space without explicitly computing the transformation.

3.5.2 Kernelized Pegasos for SVM

When kernelized, the Pegasos algorithm optimizes the SVM objective in a feature space defined by the kernel function. The optimization problem remains the same as in the linear case but is reformulated using kernel evaluations. By applying the kernelized version of Pegasos, we can effectively train SVMs on datasets that are not linearly separable, utilizing nonlinear transformations while maintaining computational efficiency.

3.6 Hyperparameter Tuning

To optimize the performance of the models before described, a step for hyperparameter tuning was performed using k-fold cross-validation. This technique splits the dataset into k (5 in this specific case) equally sized subsets (folds). The model is trained on $k-1$ folds and evaluated on the remaining fold, repeating the process k times. The average performance across all folds is used to select the best hyperparameters while preventing overfitting. A series of different parameters and the chosen model were provided as input. After generating all possible combinations of the parameters, the tuning was performed in order to identify the best parameter pair for the given model. This method was applied to tune the hyperparameters for the Perceptron, Pegasos, and Regularized logistic classification. This step was performed before and after removing the highly correlated variables. The parameters explored were:

- **Perceptron:** Learning rate η and number of iterations T.

- **Pegasos (SVM):** Regularization Parameter λ and number of iterations T.
- **Regularized Logistic Regression:** Regularization Parameter η and number of iterations T.

Here are the hyperparameters found after the removal of the variables:

- **Perceptron:** $\eta^* = 1$, $T^* = 500$.
- **Pegasos (SVM):** $\lambda^* = 1$, $T^* = 200$.
- **Regularized Logistic Regression:** $\lambda^* = 0.1$, $T^* = 200$.
- **Perceptron (Polynomial Features):**$\eta^* = 0.1$, $T^* = 200$.
- **Pegasos (SVM) (Polynomial Features):**$\lambda^* = 0.1$, $T^* = 500$.
- **Logistic (Polynomial Features):**$\lambda^* = 0.001$, $T^* = 200$.

These values yielded the lowest 0-1 loss during cross-validation, ensuring optimal model performance.

4 Performances

Now that all the algorithms have been explained and their mechanics are clear, it is possible to start analyzing the results obtained by each one of the models implemented. The evaluation is based on the 0-1 Loss, which measures how well the model's prediction match the actual labels. Mathematically, the 0-1 Loss is described as :

$$L_{0-1} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{\hat{y}_i \neq y_i\},$$

This equation indicates that the loss is the average of an indicator function which return 1 if the predicted value is different from the true value, 0 otherwise. Essentially it counts the number of misclassifications and normalizes it by the total number of samples. The smaller the 0-1 Loss the better the model implementation.

4.1 Perceptron, Pegasos and Regularized Logistic Classification

In this section it is presented the final evaluation of the models and their corresponding results based on the recently explained 0-1 Loss.

4.1.1 Given Features

After completing all the preprocessing steps, that were previously described, including selecting the right parameters and the right features, the following are the results obtained:

Model	**learning parameter**	**regularization parameter**	**Iterations**	**0-1 Loss**
Logistic	—	0.1	200	0.5805
Pegasos SVM	—	1	200	0.3109
Perceptron	1	—	500	0.3380

Table 3: Summary of the 0-1 Losses of the models and their hyperparameters.

From the data, it is immediately clear that the Perceptron Model outperforms the other models in term of 0-1 loss.

4.1.2 Polynomial Expansion Features

When polynomial feature expansion is applied we observe the following results:

Model	**learning parameter**	**regularization parameter**	**Iterations**	**0-1 Loss**
Logistic	—	0.001	200	0.5333
Pegasos SVM	—	0.1	500	0.1771
Perceptron	0.1	—	200	0.0900

Table 4: Summary of the 0-1 Losses of the models and their hyperparameters.

Even with the inclusion of polynomial features, the Perceptron still delivers the best performance in terms of 0-1 Loss. It is notable that the Perceptron's 0-1 Loss was significantly reduced, showing a considerable improvement in performance. However, the same level of improvement did not occur for the other two models (Pegasos and Logistic Classification). The Pegasos model did experience a noticeable improvement, though not substantial as the one achieved by the Perceptron. On the other hand the polynomial expansion did not bring to the Logistic classification the same benefits.

4.2 Kernelized Perceptron

Next, the performance of the Kernelized Perceptron is put under examination, focusing on the results of the two different kernels (gaussian and polynomial Kernel). The following table displays the corresponding outcome of each kernel:

Model	0-1 Loss
Gaussian Kernel	0.0624
Polynomial Kernel	0.0906

Table 5: Summary of the 0-1 Losses of the models and t.

The result suggests that the Gaussian kernel is more effective in this context, providing a better fit to the data compared to the Polynomial kernel.

4.3 Kernelized Pegasos

Finally it's the time to analyze the performance of the Kernelized Pegasos model, using, again, both kernels. The results are as follows:

Model	0-1 Loss
Gaussian Kernel	0.2122
Polynomial Kernel	0.2016

Table 6: Summary of the 0-1 Losses of the models .

Here, the polynomial kernel seems to be a better, even if slightly, choice, even though the difference in performance is relatively small.

5 Conclusion

In this study, various machine learning algorithms were explored and implemented, including the Perceptron, Support Vector Machines (SVM) using the Pegasos algorithm, and Regularized Logistic Regression, implementing them from scratch. To improve these model's performances, they were extended using polynomial feature expansion and kernel methods, enhancing their ability to capture nonlinear patterns in the data.

These experiments demonstrated that the kernelized versions of the models significantly improved classification performance, particularly when using the Gaussian kernel. The Perceptron algorithm, when kernelized, achieved the lowest 0-1 loss, indicating its strong generalization capabilities. While the perceptron always performed well, the kernelized versions outperformed those with polynomial feature expansion, while the Pegasos algorithm showed the opposite trend, performing better with the polynomial expansion than with the kernelized version. Regularized Logistic Regression had the highest 0-1 Loss, performing the worst among the models, while the pegasos algorithm, despite having good results, did not achieved the best otucomes.

Through extensive hyperparameter tuning using k-fold cross-validation, we identified optimal parameters for each model, ensuring robust performance and minimizing overfitting. Additionally, proper preprocessing, including outlier removal, feature selection, and standardization, played a crucial role in improving model performance while avoiding data leakage.

Overall, our findings highlight the effectiveness of kernel methods and regularization techniques in improving classification accuracy.