MACHINE LEARNING PROJECT

Kernelized Linear Classification

Paola Loi mat. 33088A

Università degli Studi di Milano
Data Science For Economics
A.A. 2023/2024

ABSTRACT

The goal of this project is to implement kernelized linear classification algorithms using the Python language, without relying on pre-built packages. The aim is to classify labels based on numerical features using the ℓ loss function. This is accomplished by implementing various algorithms, including the Perceptron, Support Vector Machines (SVM) with the Pegasos algorithm, and regularized logistic classification. The hyperparameters and data characteristics are compared and evaluated to prevent overfitting and underfitting, thereby identifying the most effective classification model.

Following a thorough analysis and cleaning of the dataset, the algorithms are defined structurally, and the dataset is divided into training and test sets. To enhance code clarity and flexibility, the initialization of the algorithms is deferred to a later stage within the code flow, where different hyperparameters are tested and evaluated using cross-validation. Finally, the model is tested using the optimal hyperparameters identified. The critical aspect of this project is the evaluation of algorithm performance, conducted through in-depth analysis.

Contents

1 Data Analysis and Preprocessing

The dataset contains 10 independent variables (x1, x2, ..., x10) and 1 dependent variable y, which represents a binary variable with values -1 and 1. The dataset contains 10,000 observations. The variables have different scales of values,some variables such as x3 and x10 showing a distribution of the data concentrated around specific values, while others such as x1 and x2 present a more dispersed distribution.

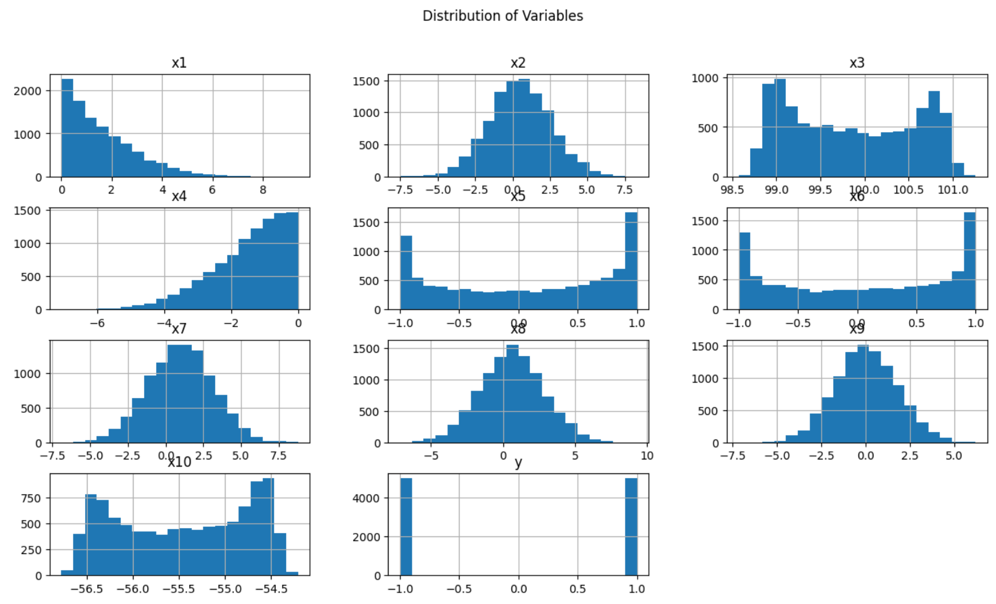


Figure 1: Variables Distribution

Looking at the distribution graphs of the variables in Figure 1, it can be immediately noticed a normal trend distribution for x2, x7 and x8. The features x1 and x4 show asymmetric distributions, with a tail on the right and left respectively. Particular attention must be paid to the variable x3 with a bimodal distribution, which could indicate the presence of two distinct subgroups in the data.

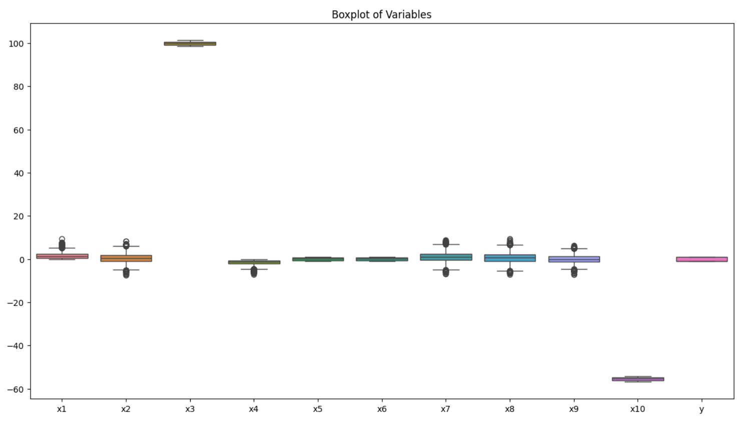
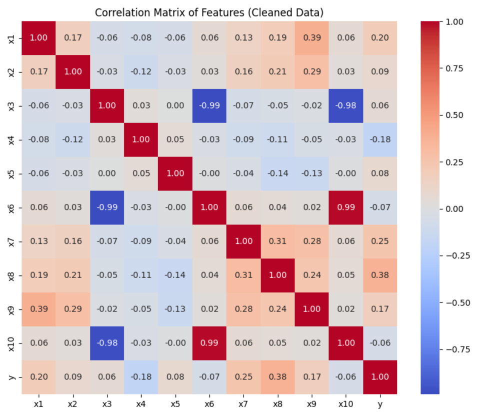


Figure 2: Variables Boxplots

Afterward, outliers are identified and removed from the variables to prevent distortions in the statistical analysis that could negatively affect the performance of the algorithms. The Boxplots in Figure 2 reveal a significant number of outliers in the variables x1, x2, x7, x8, and x10. To remove these outliers, a method based on the interquartile range (IQR) is implemented. The IQR is calculated as the difference between the third and first quartiles, and then the limits are determined, beyond which the data are considered outliers. These outlier data are subsequently eliminated.

To enhance the robustness of the dataset, a correlation analysis is conducted using a correlation matrix to facilitate the process. Particular attention is given to the variables x3, x6, and x10, which exhibit very high correlations. To mitigate potential multicollinearity issues, redundancy is addressed by excluding the variables x6 and x10 from the dataset. This step is essential for improving the stability and reliability of the machine learning models, as it reduces the risk of overfitting caused by highly correlated variables.



	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	y
x1	1.00	0.17	-0.06	-0.08	-0.06	0.06	0.13	0.19	0.39	0.06	0.20
x2	0.17	1.00	-0.03	-0.12	-0.03	0.03	0.16	0.21	0.29	0.03	0.09
x3	-0.06	-0.03	1.00	0.03	0.00	-0.99	-0.07	-0.05	-0.02	-0.98	0.06
x4	-0.08	-0.12	0.03	1.00	0.05	-0.03	-0.09	-0.11	-0.05	-0.03	-0.18
x5	-0.06	-0.03	0.00	0.05	1.00	-0.00	-0.04	-0.14	-0.13	-0.00	0.08
x6	0.06	0.03	-0.99	-0.03	-0.00	1.00	0.06	0.04	0.02	0.99	-0.07
x7	0.13	0.16	-0.07	-0.09	-0.04	0.06	1.00	0.31	0.28	0.06	0.25
x8	0.19	0.21	-0.05	-0.11	-0.14	0.04	0.31	1.00	0.24	0.05	0.38
x9	0.39	0.29	-0.02	-0.05	-0.13	0.02	0.28	0.24	1.00	0.02	0.17
x10	0.06	0.03	-0.98	-0.03	-0.00	0.99	0.06	0.05	0.02	1.00	-0.06
y	0.20	0.09	0.06	-0.18	0.08	-0.07	0.25	0.38	0.17	-0.06	1.00

Figure 3: Correlation Matrix

To avoid data leakage, the data partition and followed standardization kept a rigorous approach. Data standardization was performed exclusively on the training set. The standardization parameters obtained from the training set were then applied to the test set, ensuring that the evaluation took place in realistic conditions and without any anticipation of the test data.

2 Classifications Algorithms

2.1 The Perceptron Algorithm

The Perceptron is a linear classification algorithm that relies on an iterative process to find a hyperplane that can separate data belonging to different classes. The algorithm progressively updates the weights associated with the input features, trying to reduce the classification errors. In each iteration, if a sample is misclassified, the weights are updated to correct the error. This process continues until convergence is reached, that is, when no more classification errors occur, or until a maximum number of predetermined iterations. If the algorithm reaches convergence, it means that it has found a hyperplane that perfectly separates the classes of the dataset. Otherwise, convergence may not be achieved, indicating that the data is not linearly separable, being that the algorithm stops after a number of updates no bigger than:

$$\left(\frac{\min_{u:\gamma(u)\geq 1} \|u\|^2}{\max_{t=1,\ldots,m} \|x_t\|^2}\right) \tag{1}$$

Perceptron Convergence

In the context of Python code, the Perceptron has been implemented following this logic, starting from the definition of a class and methods.

2.1.1 Cross-Validation

In the initialization phase, the convergence of the Perceptron is tested with a different number of maximum iterations in order to identify the optimal value. This allows to understand which is the most suitable number of iterations to guarantee the convergence of the algorithm.

The cross-validation technique is used to evaluate the right combination of hyperparameters, which is implemented systematically for all the algorithms considered in the project.

The cross-validation divides the dataset into different folds, using in each iteration one of these as a test set and the rest as a training set, repeated several times. At the end of the iterations, the average accuracy of the different evaluations is calculated to determine the overall performance of the model.

The final evaluation of the algorithm on the test set is performed using only the best hyperparameter combination, selected based on the cross-validation results. The final result is expressed in terms of miss-classification rate, providing a clear measure of the accuracy of the algorithm in correctly distinguishing between the different classes present in the dataset.

2.1.2 Perceptron's results

The Perceptron output provides several key insights into the algorithm's performance with different values of maximum iterations. In particular, two values for the maximum number of iterations were tested:1000 and 10000. The cross-validation results show that, as the number of iterations increases, a marginal improvement in the mean accuracy is observed.

Max Iterations	**Mean Cross-Validation Accuracy**
1000	0.6642
10000	**0.6724**

Table 1: Mean Cross-Validation Accuracy for different Max Iterations values

Despite the improvement in accuracy with increasing iterations, the Perceptron failed to converge. This means that, even after 10000 iterations, the algorithm failed to find a set of weights that could correctly classify all the samples in the training set. The lack of convergence could be attributed to the nature of the data.

The missclassification rate obtained on the test set is 0.3337, which indicates that about 33% of the test observations were misclassified. This value suggests that the algorithm cannot optimally distinguish between the different classes present in the dataset. This result could be due to the lack of convergence or to the intrinsic complexity of the dataset, which could require a nonlinear approach for a better classification.

Finally, the final weights of the model indicate the relative importance of different features in the Perceptron's decision-making process. For example, the weight associated with the first variable is particularly high, suggesting that this variable has a significant influence on the model's decisions. In contrast, negative weights, such as those associated with variables 2 and 3, indicate an inverse correlation with the predicted output, underlining an opposite impact in the classification. It is important to note that direct interpretation of the weights can be complicated and only cursory since the model has not reached convergence, which may not accurately reflect the underlying relationships in the data.

Variable	**Final Weight**
`x1`	3.1609
`x2`	-0.7520
`x3`	-0.8585
`x4`	-0.5959
`x5`	-0.2511
`x6`	2.0813
`x7`	2.1706
`x8`	1.8178

Table 2: Final weights of the Perceptron model associated with each variable

2.2 SVMs with Pegasos algorithm

The results obtained by the Perceptron are not sufficiently satisfactory. For this reason, the next step is the implementation of an SVM algorithm, which, thanks to its ability to identify the separating plane with the maximum margin, guarantees greater robustness of the model.

The algorithm is based on the identification of the hyperplane that separates the two classes with the maximum possible distance from the data points closest to both classes. The maximum margin is defined as the maximum distance between the dataset points and the plane, allowing then to identify the optimal weight vector for classification.

The implemented Pegasos algorithm is a gradient descent method optimized to solve the primal domain problem, particularly efficient in large datasets thanks to the extraction of random samples at each iteration. Considering the result of the Perceptron algorithm we have to consider the possibility that are data are not linearly separable. Therefor we formulate the SVM to handle violations of the separation margin. The SVM objective function, then, includes a regularization term (2) and the hinge loss (3) , which penalizes misclassifications. This approach allows the model to find a balance between maximizing the margin and minimizing margin violations, making it possible to train even in the presence of complex and overlapping data.

$$\lambda \|w\|^2 \tag{2}$$

$$h_t(w) = [1 - y_t w^T x_t]_+ \tag{3}$$

During the initialization phase of the Pegasos SVM algorithm, a grid search approach was employed to explore different combinations of hyperparameters. Specifically, the regularization parameter λ and the maximum number of iterations T were varied across predefined ranges to identify the optimal values that maximize the model's performance. The cross-validation procedure was used to evaluate the mean accuracy across different folds for each combination, ensuring that the model's generalization ability is thoroughly assessed.

Lambda	**T**	**Mean Cross-Validation Accuracy**
0.001	1000	0.6146
0.001	10000	0.6813
0.001	50000	**0.7195**
0.1	1000	0.7102
0.1	10000	0.7179
0.1	50000	0.7171
1.0	1000	0.7077
1.0	10000	0.7125
1.0	50000	0.7127

Table 3: Mean Cross-Validation Accuracy for different λ and T values

2.2.1 Algorithm's results

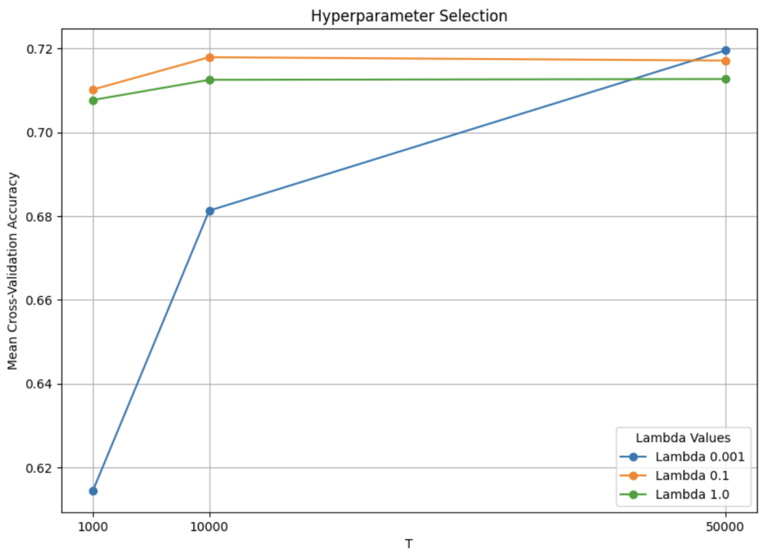


Figure 4: Hyperparameter selection

Before evaluating the model on the test set, hyperparameters are chosen through cross-validation for each combination of lambda and maximum number of iterations. The influence of T in the choice of the λ parameter is very significant. We note that by increasing T, the cross-validation accuracy tends to improve for all values of lambda, suggesting that more iterations allow the model to converge towards a more optimal solution.

In particular, for $\lambda = 0.001$, the accuracy increases significantly with the increase of T, going from 0.6146 with 1000 iterations to 0.7195 with 50000 iterations, showing the highest increase among all the tested groups. This phenomenon highlights how a weaker regularization, in this case represented by a low value of lambda, can significantly benefit from a higher number of iterations, allowing the model to better adapt to the data without incurring overfitting. The computational cost sustained to support a refinement of the prediction is in fact justified. The best parameters are T=10000 and $lambda$=0.001 with a **missclassification rate on the test test 0.2891**.

2.3 Regularized logistic classification

Again, a new algorithm is implemented in the search for a better result. Regularized logistic classification uses log-loss (4) to model the probability that an observation belongs to a class. Implemented with the Pegasos algorithm, it introduces a regularization term controlled by λ to penalize large weights, preventing overfitting and promoting generalization. Pegasos updates weights considering both log-loss and regularization, balancing the fit to training data with the ability to generalize to new data.

$$\ell(y, \hat{y}) = \log_2(1 + e^{-y\hat{y}}) \tag{4}$$

2.3.1 Algorithm's result

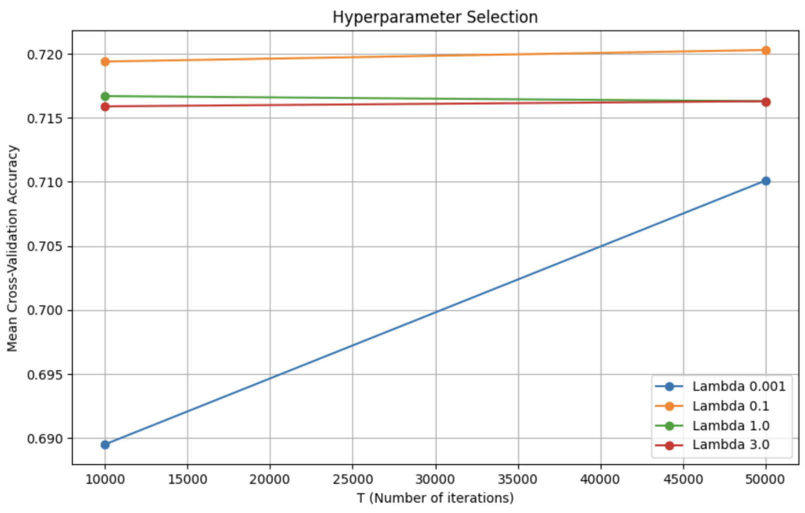


Figure 5: Hyperparameter selection

Looking at the plot, it can be noticed the rise in performance of λ=0.001 thanks to an increase in iterations from 10000 to 50000, highlighting the importance of a long T in contexts with strong regularization. For the value λ=0.1, performance does not undergo a significant increase, suggesting a robustness of the model without the need for a high number of iterations. In fact, the distance between the best parameters (λ=0.1 and T: 50000) and T 10000 is only 0.0009.

As for the values of 1.0 and 3.0, despite having a good mean-accuracy, the lines show an overlap of values and a decrease in accuracy as the iterations increase for λ=1.0, which could suggest a saturation threshold beyond which increasing the number of iterations does not bring significant benefits, probably due to an over-regularization that prevents the model from effectively adapting to the data.

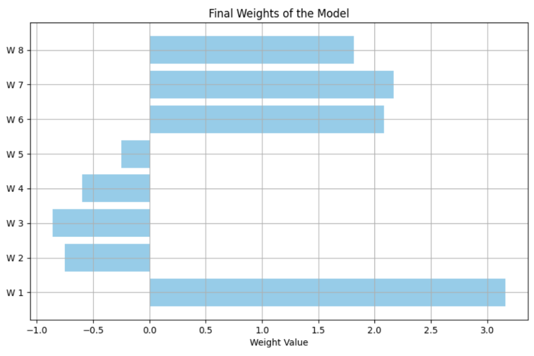
In any case, the best combination of parameters leads to a **missclassification rate on the test set of 0.3082**, a result that can certainly be improved.

3 Polynomial Expansion

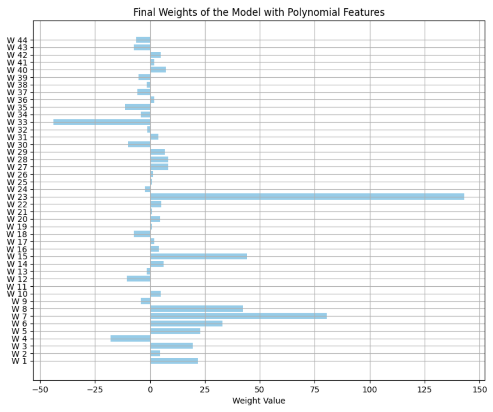
To improve the predictive capabilities, a polynomial expansion of the features is performed. This is done by extending the original space through the creation of variables made with non-linear combinations of the existing features. In this way, a non-linearity component is introduced that can be crucial in the performance of the models. In this case, we consider a polynomial expansion of degree 2. In the Python implementation, a new dataset was created that included the original and expanded features. The latter were created in such a way as to represent all the products between pairs of original features and their squares.

3.1 Perceptron's results

The analysis of the final weights in the two plots provides interesting insights into the ability of the Perceptron model, with and without the polynomial expansion, to classify the data. In the first case, the model without the polynomial expansion shows relatively uniform weights, but with modest values, except for some variables such as x1 and x7. In contrast, the model with the polynomial expansion presents a greater variability and magnitude in the weights, indicating that the model tries to exploit the introduced complexity to capture new patterns in the data and perceive novel relationships between the variables.



(a) Perceptron's Weights



(b) Polynomial expansion (Perceptron)

Figure 6: Comparison of Perceptron's Weights: Non-expanded vs. Polynomial Expansion

Although the introduction of the polynomial expansion led to an improvement in the classification at T=10000, with a cross-validation accuracy reaching the maximum value of 0.9279 and a **misclassification rate on the test set reduced to 0.0833**, the model does not fully converge. This suggests that, despite the model's efforts to adapt to a large number of interrelated features, it fails to find a solution that effectively generalizes on the test data. Some weights, such as w23 and w34, reach extremely high values, indicating that the model is trying to compensate for the lack of linear separability of the data through feature expansion. However, persistent non-convergence may indicate that the model, while fitting the training data, is unable to provide reliable predictions on new or unseen data.

3.2 Support Vector Machines' results

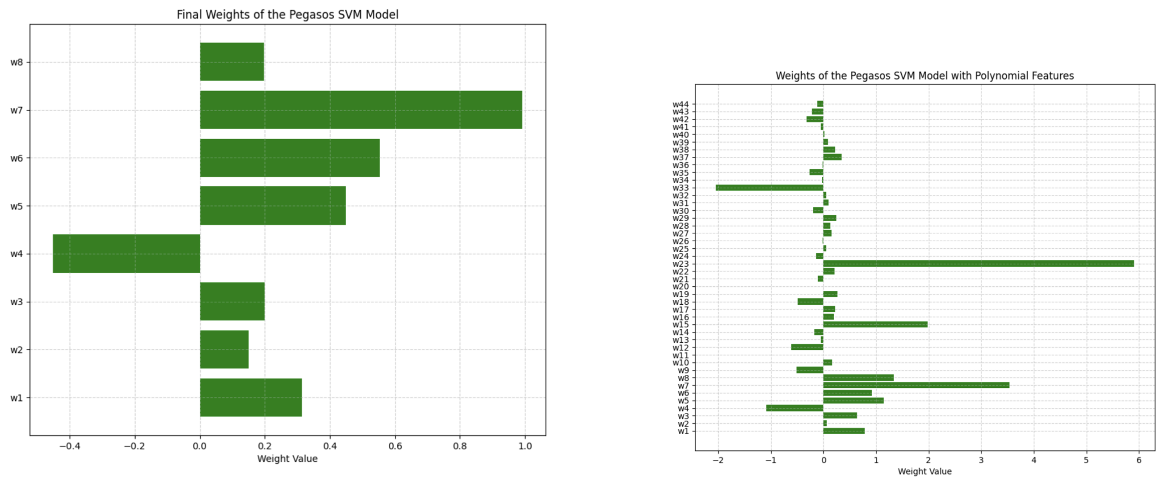
The results obtained from the implementation of the SVM with Pegasos using the second degree polynomial expansion show a significant improvement in performance compared to the linear approach without expansion. The hyperparameters that led to better results are lambda = 0.001 and T=50000, producing an accuracy with Cross-Validation equal to 0.8948. Also the **missclassification rate on the test set** has been largly improved**, being equal to 0.0727**.

Max Iter	λ **= 0.001**	λ **= 0.1**	λ **= 2.0**
10000	0.9168	0.9154	0.8747
50000	0.9340	0.9154	0.8767

Table 4: Cross-Validation Mean Accuracy

Looking at the weight plots, it is clear that in the base model, a few key variables, such as w6 and w7, dominate the decision-making process, indicating a strong dependence on specific features. With the polynomial expansion, the increase in complexity is clearly reflected in the weight distribution. Weights such as w7 and w23 show extremely high values, indicating their strong impact on the model following the polynomial expansion. These increased weights are probably the result of significant interactions between the variables that the model considers crucial for classification. On the contrary, weights such as w11 and w36 (and many other) show minimal or close to zero values, suggesting that some interactions do not make a decisive contribution to the decision-making process.

In this perspective, a handling and selection of the features to be considered could be relevant, to be counterbalanced with the excellent results on the test set.



(a) SVM's Weights

(b) Polynomial expansion (SVM)

Figure 7: Comparison of SVM's Weights: Non-expanded vs. Polynomial Expansion

3.3 Regularized logistic classification's results

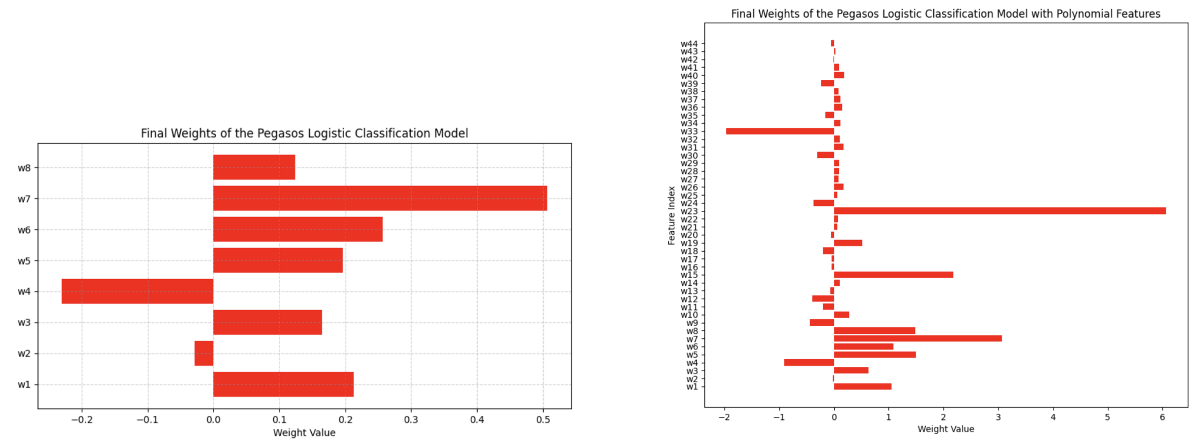
After applying the polynomial expansion to the logistic classification algorithm, significant changes can be noted, in particular in the improvements in accuracy thanks to an increase in iterations.

Afetr accuring an overflow warning a bound was added to the sigmoid function. Among the various combinations of hyperparameters, the best performance according to the evaluation via Cross-Validation on the training set has $\lambda = 0.001$ e T $= 50000$, with a value of 0.9440, indicating an accurate classification, which also are the same hyperparameter chose for the SVM algorithm, but with a lower **Misclassification rate on the test= 0.0822**

Max Iter	**Lambda 0.001**	**Lambda 1.0**	**Lambda 3.0**
10000	0.9271	0.8817	0.8340
50000	**0.9440**	0.8822	0.8438

Table 5: Cross-Validation Mean Accuracy

After the polynomial expansion, substantial differences are noted in the weights of the model. The weight w23, for example, not present in the original model, reaches the value of 6.060, indicating the importance of a new feature resulting from the interaction of two variables, demonstrating the ability of the model to capture nonlinear relationships. Considering a weight present in both models, such as w4, we observe a decrease from -0.229 in the linear model to -0.907 in the polynomial model, signaling a reduction in its influence in determining the class to which it belongs. In general, while in the linear model some weights are very significant, in the polynomial model the distribution of weights is more dispersed, suggesting a change of roles between the variables. A similarity can be noted in the distribution of weights after the polynomial expansion with the same ones of the SVM with Pegasos algorithm



(a) Logistic Classification's Weights (b) Polynomial expansion (Log Class)

Figure 8: Comparison of Logistic Classification's Weights: Non-expanded vs. Polynomial Expansion

4 Kernelized Algorithms

In order to reduce the approximation error of linear predictors, we continue with the implementation of the kernel, a function that allows to map the data from a lower input space to a higher-dimensional one. With the polynomial kernel (4), the data is mapped into a space considering all the polynomial combinations of the original features up to a certain degree. In this way, non-linear relationships can be modeled. The Gaussian kernel (5), on the other hand, transforms the data into an infinite space where the distance between the points is measured based on the Euclidean distance in the new space, thus being able to capture highly non-linear relationships.

$$K_n(x, x') = (x^\top x' + 1)^n \tag{5}$$

$$K_\gamma(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\gamma}\right) \tag{6}$$

In the implementation of the code, the algorithms' *def* are modified by adding two functions that calculate the Gaussian and polynomial kernels between samples, respectively. The introduced *support vectors* are used to update the weights through weighted sums of the kernel evaluations between the new samples and the vectors containing the missclassifications. Unlike the Perceptron, instead of keeping track of classification errors, the kernelized SVM updates the alpha parameter for the current example, increasing its value if the margin is less than 1.

4.1 The kernelized Perceptron with the Gaussian and the polynomial kernels

The Perceptron kernelized with Gaussian kernel finds the best parameter $\gamma = 0.01$, but only 10 iterations. It achieves an average accuracy on the training set of just over 82%, but its loss p of 0.1713, a relatively high value. Although the model converges in 5 epochs, the risk of overfitting remains marked given the high variability of the observed accuracy. The Perceptron with polynomial kernel, on the other hand, is set with D = 2, showing an exceptional accuracy in the cross-validation phase, equal to 95.30%. Its error rate in the test set is very low, only 0.06, but unfortunately it is not possible to reach convergence. This problem is dictated by the need to lower the maximum number of iterations, being very expensive computationally speaking, also following the numerous modifications and techniques to try to limit the computational weight. Therefore, having to think in terms of model generalization and computational speed, as well as overfitting and underfitting control, the model is not adequate to capture the complexity of the dataset used.

Table 6: CV Mean Accuracy with Gaussian kernel

Gamma	**Best CV Mean Accuracy**
0.01	0.8241
0.1	0.8241
1.0	0.8241

However, it is important to note that based on the parameters provided, the algorithm keeps the average cross-validation accuracy constant regardless of the chosen gamma value. This uniform result could indicate a lack of adequate convergence of the algorithm or suggest that the gamma parameter does not significantly influence the performance of the model in the current context. This uniformity raises questions about the effectiveness of range tuning in this specific configuration.

4.2 The kernelized Pegasos with the Gaussian and the polynomial kernels for SVM

Proceeding with the analysis of the kernelized SVM, improvements in the accuracy of the algorithm are shown in the training phase with a maximum number of 1000 iterations, both with Gaussian and polynomial kernels, the right compromise between accuracy and computational cost is found as a function of the values λ and γ in the first case and λ and polynomial degree in the second.

In the first graph, with Gaussian kernel, The graph illustrates the effect of γ values on the cross-validation accuracy for the SVM model with a number of iterations $T = 50000$. The analysis highlights that for γ 0.1, the accuracy remains substantially stable above 85%, regardless of the value of λ, showing resilience to penalty increases. On the contrary, for γ 1.0, the accuracy starts from almost 89% for low Lambda values and drops to 85% as Lambda increases, suggesting a greater sensitivity of this Gamma parameter to λ variations. The γ result was obtained with λ 0.005 and γ 1.0, reaching an accuracy of 88.47%, which demonstrates the effectiveness of combining a low λ with a high γ and a high number of iterations to improve the model performance.

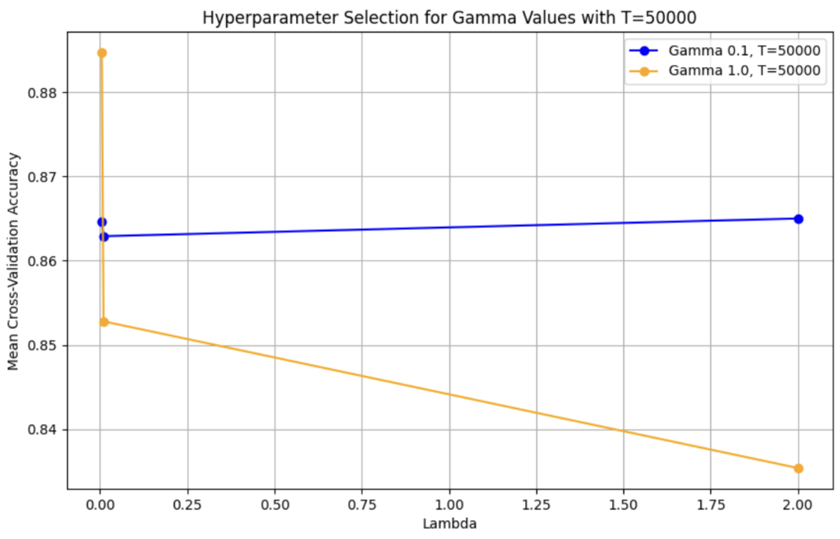


Figure 9: Gaussian Kernel

In the case of the polynomial kernel, however, a significant impact is had by the polynomial degree used with the right λ value. It can be noted that, with a polynomial degree of 2, the accuracy decreases significantly as λ increases, indicating a higher sensitivity to penalties. For degrees 3 and 4, the accuracy remains more stable and tends to improve slightly, especially for degree 4 which shows a constant increase, reaching the highest performance at λ 0.1. This suggests that a higher polynomial degree can be more effective in capturing complex relationships in the data, even with a higher λ penalty, offering an optimal balance between model complexity and generalization ability.

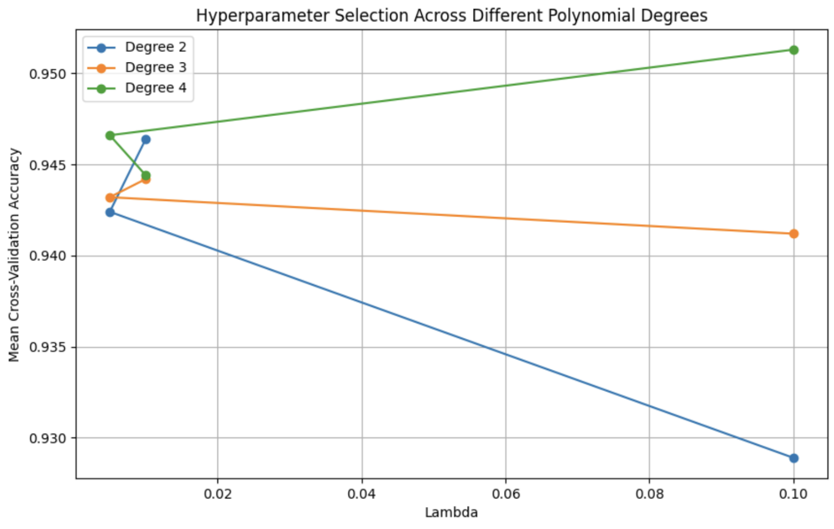


Figure 10: Polinomial Kernel

The polynomial kernel with a degree of 4 and lambda of 0.1 shows a significant superiority in both cross-validation accuracy (0.9513) and error rate on the test set (0.0578), compared to the Gaussian kernel. This shows the need for complexity to capture the relationships between non-linearly separable data. Controlled the overfitting problem,the polynomial model manages to obtain good results avoiding high computation costs and potential overfitting, increasing the complexity of the model itself.

Model	λ	**T**	γ / **d**	**CV Mean Accuracy**	**Test Misclassification Rate**
SVM Gaussian Kernel	0.005	50000	1.0	0.8847	0.11
SVM Polynomial Kernel	0.1	50000	4	0.9513	0.0578

Table 7: Comparison of SVM Gaussian and Polynomial Kernel Results

5 Conclusions

In order to evaluate the quality of the algorithms, we can observe the bar graph that shows the missclassification rate, or the loss of the algorithm in the test set.

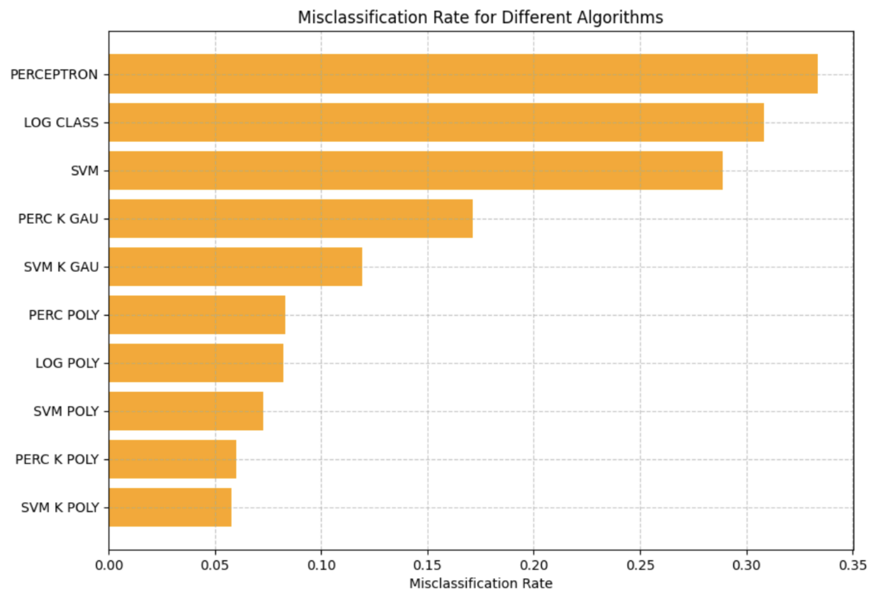


Figure 11: Algorithm Comparison

Data analysis clearly demonstrated that conventional linear models are not adequately equipped to capture the intrinsic complexity of modern datasets, resulting in suboptimal performance. Models incorporating polynomial kernels, on the other hand, demonstrated a significant decrease in misclassification rate, highlighting their superiority in modeling complex feature interactions.

However, it is essential to consider the computational implications of such models. Specifically, the kernelized perceptron, despite its success in terms of accuracy, faces significant limitations related to computational time, particularly when dealing with high polynomial degrees and large data volumes. This raises practical questions regarding its scalability and applicability in real-world scenarios.

The choice of polynomial degree 2 in the kernelized SVM with polynomial kernel shows an excellent balance between model complexity and generalization ability, suggesting that the polynomial expansion approach adopted by the Pegasos SVM could be the preferable strategy. This model, with an error rate of 0.0727, is very close to the results obtained by the polynomial kernel model, which records an error rate of 0.0578.

In conclusion, considering a trade-off between classification accuracy and computational cost, the polynomially expanded SVM model emerges as a particularly advantageous solution. This approach maintains high accuracy while remaining computationally manageable, which makes it suitable for both theoretical and practical applications.

5.0.1 APPENDIX

Down below are summarized all the algorithm ipmlemented throughout the project with their best parameters and results.

Table 8: Comparison of Perceptron Model Performances

Parameter	**Perceptron**	**Perc Expanded**	**Perc K Gaussian**	**Perc K Polynomial**
Max Iterations	10000	10000	10	10
λ / γ / **D**	-	-	$\gamma = 0.01$	Degree = 2
CV Accuracy	0.6724	0.9279	0.8241	0.9530
Misclas Rate	0.3337	0.0833	0.1713	0.06

Table 9: Comparison of SVM Model Performances

Parameter	**SVM Normal**	**SVM Expanded**	**SVM K Gaussian**	**SVM K Polynomial**
Lambda	0.001	0.001	0.005	0.1
CV Accuracy	0.7195	0.9340	0.8847	0.9513
Misclas Rate	0.2891	0.0727	0.1194	0.0578

Table 10: Comparison of Logistic Classification Model Performances

Parameter	**Logistic Normal**	**Logistic Expanded**
Lambda	0.1	0.001
CV Accuracy	0.7203	0.9440
Misclas Rate	0.3082	0.0822