Kernelized Linear Classification

Data Science for Economics - Machine Learning Project

Angelica Longo - 34314A
angelica.longo@studenti.unimi.it

February 2025

Università degli Studi di Milano Statale
A.Y. 2024/2025

I declare that this material, which 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.

Contents

1 Introduction

This project aims to implement, analyze and compare different supervised classification algorithms. The implementation must be developed in Python, entirely from scratch, avoiding libraries such as Scikit-Learn.

The project consists of the following key steps:

- An initial exploration and analysis of the dataset, focusing on understanding its structure and characteristics. To prevent data leakage, the first step is the split of the dataset in training and test sets, followed by preprocessing to handle missing values, outliers, and highly correlated variables.
- Implementation of basic classification models: the Perceptron, the Support Vector Machines (SVMs) using the Pegasos algorithm, and the Regularized Logistic Regression.
- Extension of the feature space through polynomial expansion of degree 2. The models are then re-trained and evaluated on the transformed dataset to determine whether this modification improves their performance.
- Development of kernelized models, including the Perceptron with Gaussian and Polynomial kernels, as well as the Pegasos algorithm for SVMs with both kernel types.
- A final evaluation of the results, followed by a discussion on the predictive power and computational costs of the different models, with the objective of identifying the most effective approach in this context.

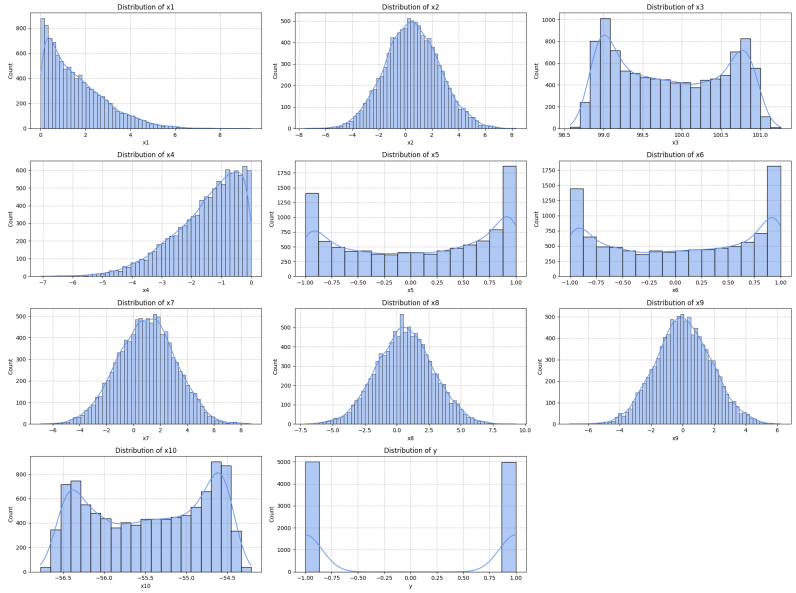
2 Data Preprocessing

After importing the required libraries and loading the dataset, preliminary adjustments are made to ensure optimal conditions for unbiased and efficient model training.

The dataset initially consists of 11 columns, including the target dependent variable y and 10 independent features $(x_1, x_2, \ldots, x_{10})$.

2.1 Variables distribution

The distribution of each variable is analyzed to understand its characteristics:



It is observed that some variables, such as x_2 and x_9, follow an approximately normal distribution. Conversely, variables like x_1 and x_4 exhibit skewed distributions, indicating a probable presence of outliers.

2.2 Handling missing Values

To avoid biased training and incorrect predictions, the dataset is checked for missing values, with the aim of removing rows with missing data. Fortunately, no missing values are found, so no imputation is required.

2.3 Data Splitting

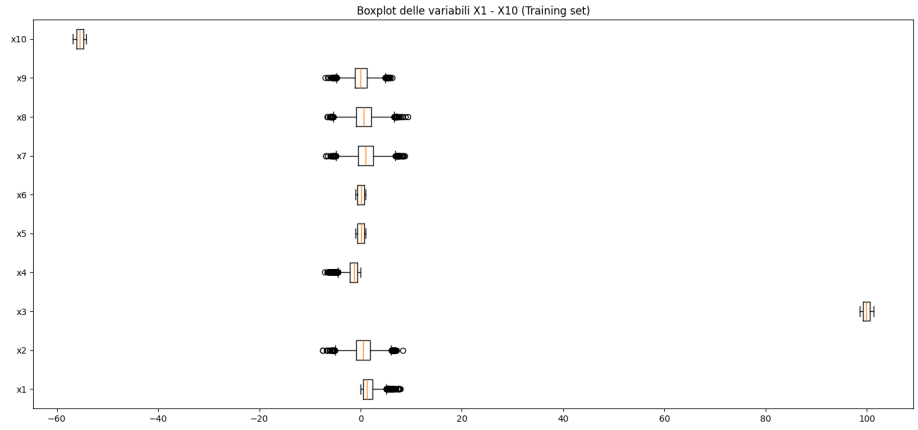
To prevent data leakage and ensure fair model evaluation, the dataset is split before any further processing, with 80% of data used as training set and 20% used as test set. Data leakage occurs when information from the test set unintentionally influences the training process, leading to overly optimistic performance estimates. So, the sight of test data should be avoided.

From this point forward, all preprocessing steps are applied only to the training set to maintain the integrity of the test data.

2.4 Outlier Detection and Removal

From the previous distribution analysis, the presence of outliers is noticeable. These extreme values, which deviate significantly from the rest of the dataset, can distort statistical analysis and negatively impact model performance.

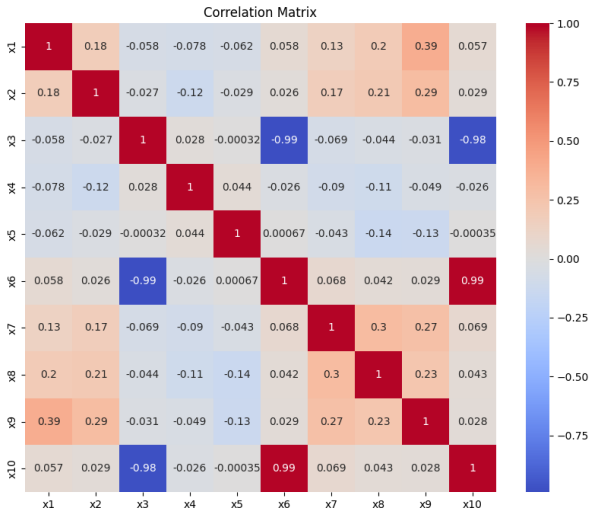
Boxplots are generated to visualize outliers:



A significant number of outliers is revealed in variables x_1, x_2, x_4, x_7, x_8 and x_9. A total of 435 outliers are detected and subsequently removed using the interquartile range (IQR) method. This slightly reduces the dataset size, but ensures a better data representation of the underlying distribution.

2.5 Correlation Analysis

The following heatmap quantifies the strength and direction of the relationships between features. Strong negative correlations are observed between x_3, x_{10}, and x_3, x_6, while strong positive correlation is observed between x_6 and x_{10}.



To mitigate the risk of multicollinearity, which may negatively impacts model performance, features x_6 and x_{10} are removed, ensuring a well-prepared dataset for modeling.

2.6 Feature Scaling

Statistics of the training set before scaling:

	x1	x2	x3	x4	x5	x7	x8	x9
count	7565.000000	7565.000000	7565.000000	7565.000000	7565.000000	7565.000000	7565.000000	7565.000000
mean	1.507794	0.470522	99.850220	-1.433707	0.080124	0.946816	0.583356	0.019144
std	1.193979	1.962075	0.711578	1.034165	0.705879	2.094444	2.110366	1.690809
min	0.002443	-4.990252	98.572455	-4.519782	-0.999999	-4.868658	-5.334468	-4.674743
25%	0.508320	-0.886841	99.160373	-2.102879	-0.618762	-0.495427	-0.888826	-1.126749
50%	1.253208	0.453024	99.805679	-1.229144	0.167511	0.985639	0.582012	-0.013010
75%	2.263779	1.825177	100.552803	-0.593386	0.776518	2.385708	2.013330	1.186029
max	5.067727	6.015575	101.260768	-0.000003	1.000000	6.806107	6.553374	4.811665

Large variations in mean values (e.g., between x_2 and x_3) indicate that features have different scales. To ensure that all features contribute equally and prevent some variables from dominating the model, standardization is performed, so that all features have zero mean and unit variance.

In order to avoid data leakage the standardization is performed only on the training set. The fitted scaler is then applied to the test set, avoiding any anticipation of the test data.

2.7 Distribution of Labels

The last step of data preprocessing is verifying the balance of target variable labels. A balanced dataset is crucial to prevent bias toward the majority class, which can lead to poor generalization. The label distribution in the training set is summarized as follows:

Class	Count (Percentage)
-1	3919 (51.80%)
1	3646 (48.20%)

Table 1: Distribution of Labels in the Training Set

The dataset is relatively balanced, with only a slight majority in class -1, allowing for fair model evaluation.

3 Algorithms Implementation

The implemented algorithms are optimized using k-fold cross-validation, which involves splitting the training set into k subsets (folds) and train the model k times, each time using $k-1$ folds for training and the remaining fold for validation. In this study, 5-fold cross-validation is used for base and expanded algorithms, to assess how well the model generalizes to unseen data and to identify the best hyperparameters. Once the optimal parameters are identified, the model is retrained on the full training set and evaluated on the test set.

The effectiveness of the prediction of the models on unseen test set is evaluated with the zero-one loss function:

$$\ell(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases}$$

which quantifies the discrepancy $l(y, \hat{y})$ between the predicted label $\hat{y}$ and the true label y. The average loss over the test dataset corresponds to the misclassification rate, which measures the proportion of incorrect predictions. By computing

$$\text{Test Accuracy} = 1 - \text{Misclassification Rate}$$

the test accuracy is obtained and can be compared to the training accuracy to assess whether the model is experiencing overfitting or underfitting.

3.1 Perceptron Algorithm

The Perceptron algorithm is one of the simplest supervised learning algorithms used for binary classification tasks. Given a set of input features, the Perceptron assigns weights to each feature and makes predictions based on a weighted sum of the inputs. It updates feature weights iteratively whenever a misclassification occurs, continuing this process for a predefined number of epochs or until convergence, meaning that no misclassifications are found.

When the predicted label $\hat{y}_t$ does not match the true label y_t

$$y_t \cdot (\mathbf{w}^\top \mathbf{x}_t) \leq 0$$

the Perceptron updates the weight vector $\mathbf{w}$ using the following rule:

$$\mathbf{w} \leftarrow \mathbf{w} + y_t \mathbf{x}_t$$

The predicted label $\hat{y}_t$ is computed using the sign function applied to the linear combination of weights and input features:

$$\hat{y}_t = \text{sign}(\mathbf{w}^\top \mathbf{x}_t)$$

The Perceptron is particularly effective when the data is linearly separable, i.e. a straight line or a hyperplane can perfectly separate the two classes. When the data is not linearly separable, the algorithm fails to converge and continues updating the weights indefinitely, leading to poor generalization and inaccurate predictions. To mitigate this, the number of epochs should be carefully chosen, to control how many times the algorithm iterates over the entire training set.

To find the optimal number of epochs, 5-fold cross-validation is performed:

Epochs	Average Accuracy	Convergency
100	0.6418	Failed
500	0.6231	Failed
1000	0.6300	Failed
2000	0.6233	Failed

Table 2: Perceptron Algorithm Accuracy

Best hyperparameters: epochs = 100
Best cross-validation accuracy: 0.6418
Misclassification rate 0.3870
Test accuracy: 0.6130

Increasing the number of epochs does not necessarily lead to improved performance. In fact, even with 5000, 10000, 20000, and 30000 epochs tested, the best configuration (5000 epochs) resulted in only a 0.01% improvement in test accuracy, adding computational complexity without significant gains in performance.

Even with 30000 epochs, the algorithm consistently reaches the maximum allowed epochs without converging, meaning that it fails to find a set of weights that correctly classifies all training samples. This suggests that the dataset is not linearly separable, which is a fundamental limitation of the Perceptron algorithm in this scenario. This highlights the need for more advanced techniques to improve model performance in non-linearly separable datasets.

Once the optimal parameters are identified, the model is retrained on the full training set and evaluated on the test set. The obtained test set's misclassification rate is 0.3870, meaning that about 39% of test observations are misclassified.

3.2 Pegasos Algorithm

The Pegasos (Primal Estimated sub-GrAdient SOlver for SVM) algorithm is an optimization-based approach for training Support Vector Machines (SVMs). Formally, an SVM seeks to identify a hyperplane that maximizes the margin between data points of different classes, thus achieving the best separation between them. This hyperplane is called the "maximally separating hyperplane", and its margin is defined as the distance between the hyperplane and the nearest data points, which are referred to as Support Vectors.

The Pegasos algorithm efficiently handles large datasets by using stochastic gradient descent (SGD) to update the weight vector iteratively based on small data subsets, reducing computational complexity. The goal of Pegasos is to minimize the SVM objective function, balancing margin maximization and misclassification penalties through hinge loss and a regularization term.

The weight update rule depends on whether a selected training sample is correctly classified:

$$w = (1 - \eta\lambda)w + \eta yx$$

where η is the learning rate, λ is the regularization parameter, y is the label and x is the feature vector. If the selected training sample is correctly classified, the update is only influenced by the regularization term, ensuring that the model does not overfit by continuously adjusting the weight vector.

In this context, it is essential to carefully select not only the maximum number of epochs T but also the values for the regularization parameter λ and the learning rate η.

The learning rate function used in this project is defined as:

$$\eta_t = \frac{1}{1+t}$$

This type of learning rate decreases inversely with the number of iterations, starting with a relatively large value and gradually decreasing as the algorithm progresses. This ensures that the updates become smaller over time, allowing for greater stability as the optimization converges.

The correct choice of λ is also critical. A high λ results in strong regularization, heavy penalty on the weights and reduction of model complexity. This can lead to underfitting, as the model may fail to capture important relationships between the variables and the targets. On the other hand, a low λ weakens the regularization, allowing the model to fit too closely to the training data, potentially leading to overfitting by capturing noise and unnecessary details.

A 5-fold cross-validation process is performed to determine the best combination of λ and T.

Iterations	λ	Average Accuracy	Iterations	λ	Average Accuracy
5000	0.001	0.7128	20000	0.001	0.7169
	0.1	0.7150		0.1	0.7134
	1	0.7150		1	0.7110
	2	0.7087		2	0.7093
10000	0.001	0.7143	30000	0.001	0.7143
	0.1	0.7139		0.1	0.7171
	1	0.7108		1	0.7126
	2	0.7110		2	0.7093

Table 3: Pegasos Algorithm Accuracy

Best hyperparameters: $\lambda = 0.1$, epochs $= 30000$
Best cross-validation accuracy: 0.7171
Misclassification rate 0.2790
Test accuracy: 0.7210

The accuracy generally remains in a small range for all the number of epochs considered. A small λ $(0,01 \text{ or } 0,1)$ tends to produce slightly higher accuracy, suggesting that stronger regularization might restrict the model too much. The best configuration achieves an accuracy of 0.7171, meaning this combination provides the best balance between regularization and model complexity.

The best-performing model has a test misclassification rate of 0.2790, resulting in a notable improvement compared to the Perceptron algorithm, demonstrating Pegasos effectiveness for complex datasets. However, the test accuracy remains quite low, meaning the model is not capturing enough of the patterns in the data.

3.3 Regularized Logistic Regression

Logistic Regression estimates the probability that a given input belongs to a particular class using the sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

with z being the weighted sum of inputs plus a bias. The sigmoid function maps the linear combination of features and weights to a value between 0 and 1, interpreted as a probability. To estimate the weights w, the logistic loss function, used instead of hinge loss chosen for SVM algorithm, is minimized. Regularization is added to this loss function to prevent overfitting by penalizing large weights, commonly using L2 regularization, which adds a term $\frac{\lambda}{2}\|\mathbf{w}\|^2$ to the loss, where λ is the regularization parameter.

A 5-fold cross-validation is performed to evaluate the accuracies for each combination of T and λ. The learning rate is the same used in the Pegasos algorithm.

Iterations	λ	Average Accuracy	Iterations	λ	Average Accuracy
5000	0.001	0.7055	20000	0.001	0.7125
	0.1	0.7098		0.1	0.7130
	1	0.7159		1	0.7121
	2	0.7105		2	0.7108
10000	0.001	0.7102	30000	0.001	0.7155
	0.1	0.7113		0.1	0.7146
	1	0.7130		1	0.7132
	2	0.7100		2	0.7113

Table 4: Logistic Algorithm Accuracy

Best hyperparameters: $\lambda = 1$, epochs $= 5000$
Best cross-validation accuracy: 0.7159
Misclassification rate 0.2885
Test accuracy: 0.7115

The optimal performance is reached with $T = 5000$ and increasing the number of iterations does not significantly improve accuracy, suggesting that the model converges effectively within this range.

3.4 Model Comparison and Brief Considerations

Model	Best Hyperparameters	Training Accuracy	Test Accuracy
Perceptron	$T = 100$	0.6418	0.6130
Pegasos	$T = 30000$, $\lambda = 0.1$	0.7171	0.7210
Logistic Regression	$T = 5000$, $\lambda = 1$	0.7159	0.7115

Table 5: Performance Comparison of Base Models

The Perceptron achieves the lowest performance, making it less favorable compared to Pegasos and Logistic Regression, that are better suited for capturing more complex relationships between features.

However, the test accuracies remain low, indicating that the models suffer from underfitting and do not generalize well to unseen data. One possible way to improve performance could be to increase the number of iterations or reduce the regularization parameter, as excessive regularization may prevent the models from fully capturing the data's complexity. However, both approaches have already been tested and have resulted in lower accuracies, suggesting an inherent issue with the data, likely due to its non-linearly separable nature. Therefore, in the following sections, feature expansion and kernel methods will be implemented to address this issue.

4 Polynomial Feature Expansion

Polynomial feature expansion is used when basic linear machine learning models are not complex enough to capture the underlying relationships in the data. It consists in expanding the original feature space through the creation of additional variables made with non-linear combinations of the existing ones. It is important to note that this method may cause overfitting, especially if the dataset contains noise or is relatively small.

For this study a polynomial feature expansion of degree 2 is applied, creating a new expanded dataset that includes all the original features, their squared terms, and their pairwise products.

Expanding the feature space in this manner allows the models to learn more complex patterns, potentially leading to improved predictive performance. To assess the actual improvement, the models are retrained using the expanded dataset and the results are analyzed in the following subsections.

4.1 Expanded Perceptron

Epochs	Average accuracy	Convergency
100	0.9169	Failed
500	0.9257	Failed
1000	0.9240	Failed
2000	0.9200	Failed

Table 6: Expanded Perceptron Accuracy

Best hyperparameters: epochs = 500
Best cross-validation accuracy: 0.9257
Misclassification rate 0.0805
Test accuracy: 0.9195

The Perceptron's training accuracy improves significantly from approximately 64% to around 92% across all epoch configurations, with 500 epochs yielding the best performance. The model clearly benefits from the data expansion, demonstrating that the introduction of non-linear features helps capture complex patterns that were previously inaccessible with a purely linear model.

However, despite the significant reduction in the test set misclassification rate and an improved test accuracy of 91.95%, the Perceptron still fails to converge. This is expected because the Perceptron does not guarantee convergence unless the data is perfectly separable. Even after polynomial expansion, the dataset might still contain overlapping classes in the new feature space, leading to continuous weight updates without reaching a stable solution.

4.2 Expanded Pegasos

Iterations	λ	**Average Accuracy**	**Iterations**	λ	**Average Accuracy**
5000	0.001	0.7881	20000	0.001	0.8378
	0.1	0.8640		0.1	0.8702
	1	0.8712		1	0.8706
	2	0.8672		2	0.8744
10000	0.001	0.8067	30000	0.001	0.8005
	0.1	0.8755		0.1	0.8890
	1	0.8677		1	0.8728
	2	0.8724		2	0.8744

Table 7: Expanded Pegasos Accuracy

Best hyperparameters: $\lambda = 0.1$, epochs = 30000
Best cross-validation accuracy: 0.8890
Misclassification rate 0.0975
Test accuracy: 0.9025

After applying polynomial feature expansion, the Pegasos algorithm shows a substantial improvement in classification accuracy. The cross-validation accuracy on the training set increases from the original range of $71 - 72\%$ to approximately $87 - 89\%$, with 88.90% being the highest

accuracy.

The test set misclassification rate decreases to 0.0975, yielding a final test accuracy of 90.25%, demonstrating that the polynomial expansion allows Pegasos to refine the decision boundary, leading to better generalization on unseen data.

4.3 Expanded Logistic

Iterations	λ	Average Accuracy	Iterations	λ	Average Accuracy
5000	0.001	0.8295	20000	0.001	0.8402
	0.1	0.8490		0.1	0.8806
	1	0.8773		1	0.8792
	2	0.8616		2	0.8611
10000	0.001	0.8410	30000	0.001	0.8102
	0.1	0.8689		0.1	0.8735
	1	0.8794		1	0.8818
	2	0.8575		2	0.8591

Table 8: Exapanded Logistic Accuracy

Best hyperparameters: $\lambda = 1$, epochs $= 30000$
Best cross-validation accuracy: 0.8818
Misclassification rate 0.1195
Test accuracy: 0.8805

For Logistic Regression, the introduction of polynomial features also results in a significant performance boost. The cross-validation accuracy on the training set increases from $71 - 72\%$ to approximately $86 - 88\%$, with the best-performing configuration achieving a cross-validation accuracy of 88.18%. The test set misclassification rate decreases to 0.1195, corresponding to a final test accuracy of 88.05%.

4.4 Model Comparison and Brief Considerations

Model	Best Hyperparameters	Training Accuracy	Test Accuracy
Perceptron	$T = 500$	0.9257	0.9195
Pegasos	$T = 30000$, $\lambda = 0.1$	0.8890	0.9025
Logistic Regression	$T = 30000$, $\lambda = 1$	0.8818	0.8805

Table 9: Performance Comparison of Expanded Models

The polynomial feature expansion significantly improves model performance, demonstrating that a non-linear transformation of the input space can enhance classification accuracy. However, computational cost and risk of overfitting must be carefully managed, making regularization and iteration control crucial.

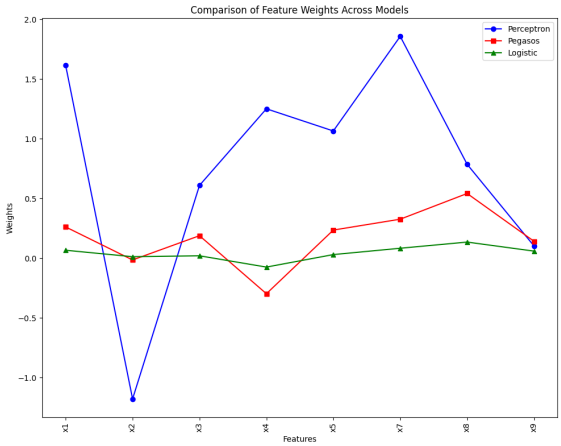
For the expanded models, there is no sign of overfitting or underfitting: all models perform well, with the expanded Perceptron achieving the highest test accuracy.

5 Weights Analysis and Comparison

In this section linear weights associated with both the original numerical features and the transformed features resulting from polynomial expansion are compared. This analysis aims to under-

stand the relative importance of each feature in the classification process for each model. The variables with the highest absolute weight values contribute the most to classification decisions. In contrast, the negative weight of a feature indicates its inverse correlation with the predicted output.

5.1 Weights Comparison of Base Models



The weights assigned to each feature vary significantly across algorithms, reflecting their different optimization strategies and decision boundaries.

The high variability in the Perceptron's weights suggests difficulties in convergence, meaning the weights may not accurately reflect the true feature-label relationships. For both Pegasos and Logistic, features have more balanced weight distributions, meaning that they contribute fairly evenly to model's classification decisions and that these models rely on multiple features rather than overemphasizing a few.

5.2 Weights Comparison of Expanded Models

The following table shows the relative importance of each non-linear feature introduced by polynomial expansion. Only some interactions are reported, but still some considerations can be made.

Var	Perceptron	Pegasos	Logistic
x_1	26.8908	0.3559	0.0633
x_2	-3.1774	0.0360	0.0206
x_3	11.0321	0.1886	0.0262
x_4	-17.7072	-0.2548	-0.0610
x_5	33.2033	0.3024	0.0392
x_7	23.9675	0.2635	0.0867
x_8	68.4238	0.7064	0.1431
x_9	26.4281	0.2685	0.0574
x_1^2	-6.8570	-0.1831	0.0046
x_2^2	-1.5422	0.1396	0.0601
x_3^2	-3.1127	-0.1299	-0.0288
x_4^2	-8.6679	-0.0818	-0.0066
x_5^2	-12.1799	-0.0102	-0.0330
x_7^2	-10.3294	-0.0642	-0.0240
x_8^2	-6.2222	-0.0294	-0.0139
x_9^2	3.0898	0.0952	0.0334

Var	Perceptron	Pegasos	Logistic
$x_1 * x_2$	1.3543	0.1418	0.0542
$x_1 * x_3$	1.0231	-0.0756	-0.0063
$x_1 * x_4$	-14.3259	-0.1508	-0.0182
$x_1 * x_5$	-0.7552	-0.0663	-0.0211
$x_1 * x_7$	-0.2835	-0.0322	0.0141
$x_1 * x_8$	43.3280	0.3112	0.0541
$x_1 * x_9$	-3.4649	0.0611	0.0145
$x_2 * x_3$	-4.4911	-0.0414	-0.0144
$x_2 * x_4$	5.0618	0.0633	-0.0052
$x_2 * x_5$	1.4147	-0.0667	-0.0227
$x_2 * x_7$	-0.2434	0.0450	0.0278
$x_2 * x_8$	7.3917	-0.0467	0.0213
$x_2 * x_9$	128.6733	0.9504	0.1856
$x_3 * x_4$	0.0209	-0.0237	0.0070
$x_3 * x_5$	7.3597	0.0797	0.0102
$x_3 * x_7$	-3.7019	0.0099	0.0050

Table 10: Expanded Algorithms Weights Comparison

Notably, the Perceptron exhibits extreme weight magnitudes, particularly for interaction terms like $x_2 \cdot x_9$ and $x_1 \cdot x_8$, suggesting heightened sensitivity to these specific feature combinations.

This confirms that the Perceptron, lacking regularization, is prone to amplifying certain feature interactions excessively, leading to potential overfitting. Pegasos and Logistic Regression distribute weights more evenly, suggesting that regularization controls the impact of individual features and their interactions, preventing any single feature from dominating the decision process.

6 Kernelized Models

Kernel models provide a powerful way to enhance predictive performance by transforming data into higher-dimensional spaces, where linear separation may become feasible. The key idea behind kernel models is to map the input data into a new feature space, allowing the model to capture complex patterns and relationships more effectively. At the heart of this transformation process are kernel functions, which compute dot products between transformed data points in the higher-dimensional space without explicitly performing the transformation. This technique, known as the kernel trick, allows models to efficiently handle complex datasets while maintaining computational feasibility.

In this chapter, the Perceptron and Pegasos algorithms using the polynomial kernel and the Gaussian (RBF) kernel are implemented.

The polynomial kernel captures nonlinear relationships by computing dot products in a polynomially expanded feature space:

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^d$$

where d is the polynomial degree, which determines the complexity of the feature interactions modeled by the kernel. Higher values of d allow for more complex decision boundaries, at the cost of increased computational complexity.

The Gaussian kernel, or Radial Basis Function (RBF) kernel, measures similarity between data points based on their distance in the original input space:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma}\right)$$

where σ controls the width of the kernel and influences the model's sensitivity to local structures in the data. A smaller σ results in a more flexible model that captures fine-grained variations, while a larger σ smooths decision boundaries.

To balance computational efficiency with robust model selection, 2-fold cross-validation is used instead of the 5-fold approach used in previous sections.

6.1 Kernelized Perceptron

The kernelized Perceptron algorithm extends the classical Perceptron by incorporating a kernel function K, allowing it to learn non-linear decision boundaries. The decision function is given by:

$$h(x) = \text{sgn}\left(\sum_{s \in S} y_s K(x_s, x)\right)$$

where S is the set of support vectors, y_s are their corresponding labels and $K(x_s, x)$ is the kernel function applied to the support vectors and the input x.

Although the kernelized Perceptron provides greater flexibility for modeling non-linear decision boundaries, it requires careful selection of the kernel function and its parameters (e.g., σ for the Gaussian kernel, degree for the polynomial kernel). This makes it computationally expensive, especially for large datasets, where convergence can be difficult to achieve.

6.1.1 Gaussian Kernelized Perceptron

First, the Gaussian kernel is applied and a 2-fold cross-validation is used to tune the hyperparameter σ and the number of epochs T.

Epochs	Sigma	Accuracy	Convergency (k_1, k_2)
10	0.1	0.7964	5 epochs, 6 epochs
	0.5	0.8434	5 epochs, 5 epochs
	1	0.9225	Failed, Failed
	2	0.9461	Failed, Failed
15	0.1	0.7964	5 epochs, 6 epochs
	0.5	0.8434	5 epochs, 5 epochs
	1	0.9240	Failed, 15 epochs
	2	0.9523	Failed, Failed
20	0.1	0.7964	5 epochs, 6 epochs
	0.5	0.8434	5 epochs, 5 epochs
	1	0.9240	20 epochs, 15 epochs
	0.5	0.9488	Failed, Failed

Table 11: Gaussian Kernelized Perceptron Accuracy

Best hyperparameters: $\sigma = 2$, epochs $= 15$
Best cross-validation accuracy: 0.9523
Misclassification rate 0.0585
Test accuracy: 0.9415

Even with reduced computational complexity, the Perceptron still converges during parameter tuning, though the number of required epochs varied depending on the hyperparameter settings.

The results show that the performance of the Gaussian Kernelized Perceptron is significantly influenced by the choice of the hyperparameter σ, suggesting a trade-off between model complexity and convergence stability. Higher values of σ improve generalization but make convergence more challenging; lower values of σ result in faster convergence but lower overall accuracy.

Using the best-found parameters the Perceptron is retrained on the entire training set: the model fails to converges in the specified epochs limit, however it reaches an optimal test accuracy of 0.9415.

6.1.2 Polynomial Kernelized Perceptron

In addition to the Gaussian kernel, polynomial kernel is implemented, aiming to determine the optimal degree, constant and number of epochs.

Epochs	Degree	Constant	Accuracy	Convergency (k_1, k_2)
10	2	0	0.7071	Failed, Failed
		1	0.9212	Failed, Failed
	3	0	0.6775	Failed, Failed
		1	0.9401	Failed, Failed
20	2	0	0.6966	Failed, Failed
		1	0.9221	Failed, Failed
	3	0	0.6783	Failed, Failed
		1	0.9499	Failed, Failed

Table 12: Polynomial Kernelized Perceptron Accuracy

Best hyperparameters: degree = 3, constant = 1, epochs = 20
Best cross-validation accuracy: 0.9499
Misclassification rate 0.0405
Test accuracy: 0.9595

The results show that the performance of the Polynomial Kernelized Perceptron is highly sensitive to the choice of hyperparameters: a non-zero constant allows a flexible offset that helps improve generalization; a higher-degree leads to better decision boundaries. However, this also introduces stability issues, as indicated by the lack of convergence in all tested configurations.

This trade-off between model complexity and stability suggests the need for further tuning, such as increasing epoch limits. However, due to computational complexity, these additional experiments were not conducted.

In computing the test accuracy, like the Gaussian kernel, the polynomial kernel does not converge within the maximum number of epochs specified, but also in this case there is an improvement compared to the basic and expanded linear Perceptron algorithm, with a test set accuracy of 0.9595.

6.2 Kernelized Pegasos

Following the discussion on kernel methods and the kernelized Perceptron, this section explores the adaptation of the Pegasos algorithm to leverage kernel functions.

The Pegasos algorithm updates the weight vector w_{t+1} iteratively using a randomly selected subset of the dataset. At each iteration t, the update rule is given by:

$$w_{t+1} = \frac{1}{\lambda t} \sum_{i=1}^{t} \mathbf{1} \left[y_i \sum_{j=1}^{m} \alpha_t[j] y_j k(x_j, x_i) < 1 \right] y_i \phi(x_i),$$

where $\mathbf{1}(\cdot)$ is the indicator function, which evaluates to 1 if the condition holds and 0 otherwise. The weight w_{t+1} can be represented as the vector α_{t+1}, where each component $\alpha_{t+1}[j]$ tracks how many times sample j has been selected with a nonzero loss. Formally,

$$\alpha_{t+1}[j] = \left| \left\{ t' \leq t : i_{t'} = j \wedge y_j \sum_{k=1}^{m} \alpha_{t'}[k] y_k k(x_k, x_j) < 1 \right\} \right|.$$

Thus, it can be written w_{t+1} as:

$$w_{t+1} = \frac{1}{\lambda t} \sum_{j=1}^{m} \alpha_{t+1}[j] y_j \phi(x_j).$$

This dual representation allows Pegasos to efficiently handle kernelized data without explicitly computing high-dimensional weight vectors. The kernelized adaptation of Pegasos retains the algorithm's efficiency while enabling the use of non-linear decision boundaries, making it a powerful approach for complex learning tasks.

6.2.1 Gaussian Kernelized Pegasos

As for the Perceptron, first the Gaussian kernel is applied and a 2-fold cross-validation is used to tune the hyperparameter σ, λ and the number of epochs T.

Epochs	λ	Sigma	Accuracy
10	0.01	0.1	0.6730
		0.5	0.6613
		1	0.6467
		2	0.5668
	0.1	0.1	0.6019
		0.5	0.6419
		1	0.6172
		2	0.6165
	1	0.1	0.6656
		0.5	0.6024
		1	0.6835
		2	0.6295

Epochs	λ	Sigma	Accuracy
20	0.01	0.1	0.6596
		0.5	0.6597
		1	0.6460
		2	0.6214
	0.1	0.1	0.6658
		0.5	0.6987
		1	0.6664
		2	0.6172
	1	0.1	0.6768
		0.5	0.6148
		1	0.6377
		2	0.6059

Table 13: Gaussian Kernelized Pegasos Accuracy

Best hyperparameters: $\lambda = 0.1$, sigma $= 0.5$, epochs $= 20$
Best cross-validation accuracy: 0.6987
Misclassification rate 0.3690
Test accuracy: 0.6310

The Gaussian kernelized Pegasos reaches a very low training accuracy, and consequently a very low test accuracy, indicating the model does not perform well and consequently does not generalize well on unseen data.

6.2.2 Polynomial Kernelized Pegasos

Epochs	Degree	Constant	λ	Accuracy
100	2	0	0.01	0.5873
			0.1	0.5987
			1	0.5890
		1	0.01	0.6221
			0.1	0.6863
			1	0.6340
	3	0	0.01	0.6400
			0.1	0.5919
			1	0.6256
		1	0.01	0.6543
			0.1	0.6731
			1	0.6502

Epochs	Degree	Constant	λ	Accuracy
200	2	0	0.01	0.5880
			0.1	0.6163
			1	0.6186
		1	0.01	0.6348
			0.1	0.6827
			1	0.6185
	3	0	0.01	0.6444
			0.1	0.6619
			1	0.6190
		1	0.01	0.6689
			0.1	0.6884
			1	0.6830

Table 14: Polynomial Kernelized Pegasos Accuracy

Best hyperparameters: $\lambda = 0.1$, degree $= 3$, constant $= 1$, epochs $= 200$
Best cross-validation accuracy: 0.6884
Misclassification rate 0.3190
Test accuracy: 0.6810

In most cases, increasing from 100 to 200 leads to higher accuracy, suggesting that the model benefits from additional training time. One of the most evident findings is the importance of including a bias term, represented by the constant. Across different configurations, models with a nonzero constant generally perform better, suggesting that allowing a flexible offset in the decision boundary helps improve generalization.

6.3 Model Comparison and Brief Considerations

Model	Best Hyperparameters	Training Accuracy	Test Accuracy
GK Perceptron	$T = 15$, $\sigma = 2$	0.9523	0.9415
PK Perceptron	$T = 20$, degree $= 3$,constant $= 1$	0.9499	0.9595
GK Pegasos	$T = 20$, $\lambda = 0.1, \sigma = 0.5$	0.6987	0.6130
GP Pegasos	$T = 200$, $\lambda = 1$, degree $= 3$, constant $= 1$	0.6884	0.6810

Table 15: Performance Comparison of Kernelized Models

Among the proposed models, the Polynomial Kernel Perceptron achieves a very high test accuracy, making it an optimal choice for classification. However, its computational cost is excessively high. In fact, experiments with more than 15 epochs were not feasible due to the long runtimes required to compute the best hyperparameters, even with 2-fold cross-validation. For this reason, among the kernelized models, the Gaussian Kernel Perceptron is preferred, as it delivers competitive accuracy in significantly less time.

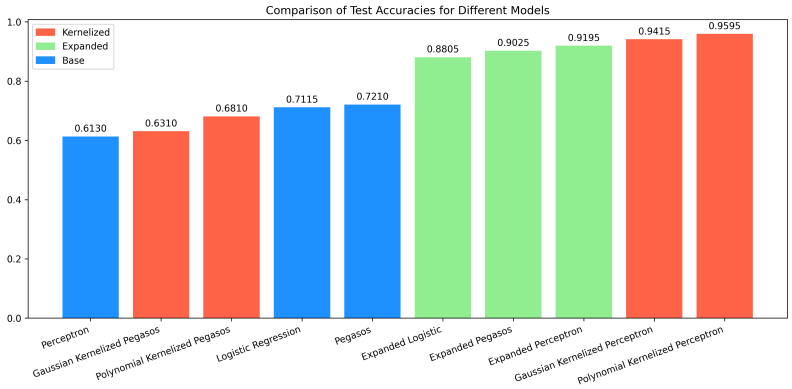
In contrast, the kernelized Pegasos algorithm achieves a test accuracy of around 0.7, significantly lower than the Perceptron models. The Pegasos algorithm involves regularization, which makes it more sensitive to hyperparameter tuning. These models likely suffer from underfitting, as the limited number of training epochs may not have been sufficient to fully explore the parameter space and optimize the model.

Overall, the flexibility of the Perceptron provides a significant advantage in this study.

7 Conclusions

The goal of this project was to classify data based on numerical feature.
For the proposed dataset, the simple structures of Perceptron, Pegasos and Logistic Regression models does not perform well, since they are often constrained by the assumption of linear separability and struggle in capturing the data complexities. To address this limitation, expanded versions and kernel methods are implemented. The expanded models enrich the feature space by introducing higher-order iterations. The kernel methods, particularly the Gaussian and polynomial kernels, further enhance model performance by mapping the data into higher-dimensional spaces where non-linear relationships could be more effectively modeled. However, it should be taken into account that, due to computational complexities, the base and expanded models were trained with a significantly higher number of epochs and used 5-fold cross-validation, whereas the kernelized models used only 2-fold cross-validation and a much lower number of epochs. These differences in the training process can significantly impact the results, making the comparison less fair. Further experiments with more folds or epochs should have been conducted, but this was not done due to the computational time required.

However, despite these limitations, some conclusions can still be drawn.



The comparison of test accuracies across various models highlights notable differences in performance. It is clearly visible that algorithms using dataset expansion through feature expansion lead to a significant improvement in performance, with all test accuracies being slightly under or above 90%, highlighting their superiority in modeling complex feature interactions.

Kernelized Pegasos' results are not satisfactory, but their reliability is questionable since their low test accuracy could be due to underfitting, caused by insufficient iterations and a limited cross-validation setup, leading to non-exhaustive training.

On the other hand, kernelized Perceptron models achieve a very high accuracy (> 0.94) even with a limited number of epochs and folds, showcasing the power of kernel methods for capturing non-linear relationships in the data. However, it is essential to consider the computational implications of such models.

The table below reports the test accuracies and computational times required to compute the best hyperparameters using cross-validation for different algorithms.

Model	**Test Accuracy**	**Computational Efficiency**
Perceptron	0.6130	839.07 seconds
Pegasos	0.7210	39.66 seconds
Logistic	0.7115	17.45 seconds
Expanded Perceptron	0.9195	222.04 seconds
Expanded Pegasos	0.9025	14.33 seconds
Expanded Logistic	0.8805	18.17 seconds
GK Perceptron	0.9415	6984.76 seconds
PK Perceptron	0.9500	21483.68 seconds
GK Pegasos	0.6310	11707.46 seconds
PK Pegasos	0.6810	2508.28 seconds

Table 16: Test Accuracy and Computational Efficiency for Different Models

Based on the test accuracy results, the best outcomes are achieved by the Gaussian and Polynomial kernelized Perceptron. But, despite their success in terms of accuracy, the models face significant limitations related to computational time, particularly when dealing with high polynomial degrees and large data volumes. This raises practical questions regarding their applicability in real-world scenarios.

Considering the trade-off between classification accuracy and computational cost, the polynomially expanded Perceptron and Pegasos models emerge as particularly advantageous solutions, as they reaches competitive accuracy while requiring significantly less time, striking a good balance between performance and computational costs. This makes them suitable for both theoretical and practical applications.