Kernelized Linear Classification

Name: Samet
Surname: CAGAN
Matriculation number: 985869
Department: Political, Economic and Social Sciences
Programme: Data Science for Economics (DSE)
Github: github.com/sametcagan/ML-project

Abstract

This study explores the application of various machine learning algorithms for binary classification on a dataset consisting of 10,000 entries with 10 numerical features. The dataset underwent comprehensive preprocessing, including feature scaling and outlier removal, to optimize it for model training. Three fundamental models—Perceptron, Pegasos SVM, and Pegasos Logistic Regression—were initially implemented, achieving modest accuracies. To address the limitations of these linear models, polynomial feature expansion was introduced, significantly improving accuracy. Furthermore, a Kernelized Perceptron with an RBF kernel was developed, achieving the highest accuracy of 95.50% after extensive hyperparameter tuning. This research highlights the critical importance of data preprocessing, feature engineering, and the use of advanced modeling techniques to enhance predictive performance in machine learning tasks.

1. INTRODUCTION

This study focuses on a binary classification problem using a dataset composed of 10,000 entries, each with 10 numerical features and a binary target variable. The dataset is well-structured, without missing values, but presents a mixture of positive and negative values across its features, suggesting that careful preprocessing is necessary. Feature scaling, outlier removal, and data visualization are integral steps to ensure that the dataset is optimally prepared for model training.

Three fundamental machine learning algorithms—the Perceptron, Support Vector Machine (SVM) using the Pegasos algorithm, and Regularized Logistic Regression—were initially implemented and evaluated. These models were chosen for their simplicity and effectiveness in various classification tasks, providing a strong foundation for further experimentation. Recognizing the potential limitations of linear models, advanced techniques such as polynomial feature expansion and kernel methods were subsequently applied to enhance the models' capacity to capture non-linear patterns within the data.

The primary objective of this study is to systematically explore and compare these machine learning models' performances, from basic linear classifiers to more sophisticated kernelized versions, ultimately identifying the most effective approach for this binary classification task. Through this process, the study aims to demonstrate the importance of data preprocessing, feature engineering, and model selection in achieving high predictive accuracy and reliability in machine learning applications.

2. DATASET EXPLORATION AND PREPROCESSING

2.1. Dataset Overview

The dataset comprises 10,000 entries, each characterized by 10 numerical features (x_1 to x_10) and a binary target variable y, where y can take values of 1 or -1. The dataset appears to be well-structured, with no missing values, as confirmed by checking the dataset's information and performing an `.isnull().sum()` operation, which returned zero for all columns.

This dataset reveals a mixture of positive and negative values across the features. The feature scaling might be necessary to ensure that the algorithms perform optimally. The target variable is balanced between the two classes.

2.2.Data Preprocessing

2.2.1. Feature Scaling

Feature scaling is a crucial step in preparing the data for kernelized methods and algorithms like SVMs and logistic regression. Initially, the dataset's features were scaled using two different methods:

2.2.1.1.StandardScaler:

This method was applied to transform the features such that each feature has a mean of 0 and a standard deviation of 1. After scaling, the features were examined using the `.describe()` method, which confirmed the successful normalization of the data.

2.3.1.2. RobustScaler:

To handle potential outliers in the data, RobustScaler was also used. This method scales features according to the interquartile range (IQR), which is less sensitive to outliers than the mean and standard deviation. The result of this scaling process was again checked, and the scaled features were stored for further use.

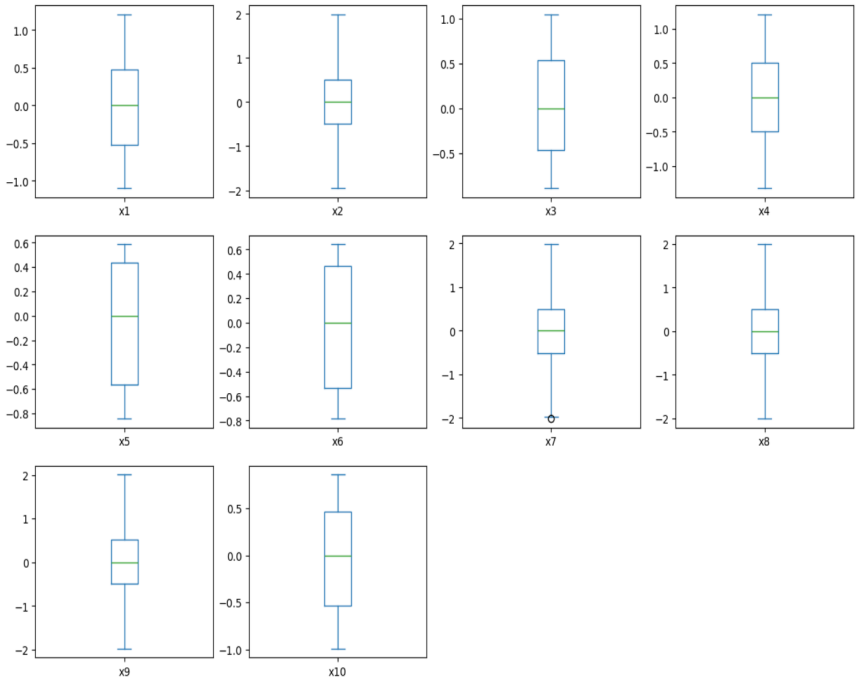
2.2.2. Outlier Removal

In the data preprocessing phase, I addressed the presence of outliers across various features, which were identified using boxplot visualizations and skewness analysis. Initially, several features exhibited significant skewness and outliers, potentially impacting model performance. To mitigate this, Iapplied the Interquartile Range (IQR) method to systematically remove outliers from all features. This process resulted in a dataset with more symmetric and normalized distributions, as evidenced by the reduced skewness values across all features and the absence of extreme values in the updated boxplots. By removing these outliers, Iaimed to reduce noise and improve the robustness of my machine learning models.

3. DATA VISUALIZATION

To further understand the distribution of the features and to inspect the impact of the scaling:

Feature	Skewness
x3	0.1019
x2	0.0340
x9	0.0233
x1	0.0214
x8	0.0053
x4	-0.0069
x7	-0.0450
x6	-0.1015
x10	-0.1053
x5	-0.1866



On the figure, most features exhibit a symmetrical distribution with few or no outliers, as seen by the absence of dots outside the whiskers in most of the plots. This indicates that the data is well-prepared for further analysis and modeling, with reduced skewness and a more consistent range of values across features.

On the table, skewness values for each feature after the outlier removal and transformation processes. Skewness is a measure of the asymmetry of the data distribution. This value close to zero indicates that the distribution is nearly symmetrical. The table shows that all features values close to zero.

4. MODEL IMPLEMENTATION

In this section, I implemented three fundamental machine learning algorithms from scratch:

1. Perceptron
2. Support Vector Machine (SVM) using the Pegasos algorithm
3. Regularized Logistic Regression using the Pegasos algorithm

Each model was tested on the preprocessed dataset to evaluate its performance.

4.1. Perceptron

The Perceptron algorithm aims to find a linear decision boundary that separates data into different classes by iteratively adjusting its weights and bias based on the input features and corresponding labels.

4.1.1. Model Implementation:

The Perceptron was initialized with a learning rate of 0.01 and set to iterate through the training data 1,000 times. The learning rate determines the size of the steps the algorithm takes during each iteration to update the weights, while the number of iterations dictates how many times the algorithm processes the entire training dataset.
During the training phase, the model starts with the weights and bias initialized to zero. For each data sample, the Perceptron calculates a weighted sum (linear output) of the input features and predicts the class label using the sign function. If the predicted label differs from the actual label, the model updates the weights and bias to reduce the error. This process is repeated for each sample across all iterations.
After training, the model was used to make predictions on the training dataset. The Perceptron applies the learned weights and bias to each sample, computes the linear output, and then predicts the class based on the sign of this output.

4.1.2. Model Performance:

The Perceptron model achieved an accuracy of 67.61% on the training dataset. While this indicates that the model has learned some patterns from the data, it may not be capturing all the important relationships. This level of accuracy suggests the possibility of underfitting, where the model is too simplistic to fully represent the underlying data patterns. In particular, as the Perceptron is a linear classifier, underfitting can occur if the data contains complex patterns or non-linear relationships that a simple linear model cannot adequately capture.

4.2. Support Vector Machines (SVMs) Using the Pegasos Algorithm:

The Pegasos algorithm is an efficient method for training Support Vector Machines (SVMs), particularly on large datasets. The algorithm iteratively adjusts the model's weights and bias to maximize the margin between classes while minimizing the classification error. It is well-suited for large-scale machine learning tasks due to its simplicity and effectiveness.

4.2.1. Model Implementation:

The Pegasos SVM was initialized with a regularization parameter lambda_param of 0.01 and set to run for 1,000 iterations. The regularization parameter controls the trade-off between achieving a low error on the training data and maintaining a large margin. The algorithm

initializes the weights and bias to zero and iteratively updates them based on randomly selected samples from the training data.
During each iteration, the algorithm selects a random sample and computes the learning rate, which decreases over time as the number of iterations increases. The model checks whether the selected sample violates the margin requirement (i.e., whether it is misclassified or within the margin). If a violation occurs, the weights and bias are updated to correct the error; otherwise, the weights are slightly adjusted to maintain regularization.
After training, the Pegasos SVM was used to predict class labels for the test dataset by applying the learned weights and bias to each sample and computing the decision boundary.

4.2.2. Model Performance:

The Pegasos SVM model achieved an accuracy of **73.17%** on the test dataset. This suggests that the model has effectively captured many patterns in the data, improving on the Perceptron's performance. However, since the model relies on a linear decision boundary, there is still a possibility of underfitting if the data contains non-linear relationships that the model cannot capture.

4.3. Pegasos Logistic Regression

The Pegasos Logistic Regression model combines the efficiency of the Pegasos algorithm with the logistic loss function, making it well-suited for binary classification tasks. Unlike linear SVMs, this model is designed to output probabilities, providing a more nuanced prediction by estimating the likelihood of a data point belonging to a particular class.

4.3.1. Model Implementation:

The Pegasos Logistic Regression model was initialized with a regularization parameter lambda_param of 0.01 and a maximum of 1,000 iterations. The model starts by initializing weights and bias to zero. During each iteration, the algorithm selects a random sample from the training data and calculates a learning rate that decreases over time.
For each selected sample, the model computes the margin, which is the product of the sample's features and the current weights, plus the bias. The logistic loss gradient is then calculated using the sigmoid function, which helps to update the weights and bias gradually, ensuring that the model minimizes the logistic loss over time.
After training, the model was used to predict class labels for the test dataset. Predictions are made by applying the learned weights and bias to each sample and using the sigmoid function to determine the probability of class membership.

4.3.2. Model Performance:

The Pegasos Logistic Regression model achieved an accuracy of **70.40%** on the test dataset. While this accuracy suggests that the model has effectively captured a significant portion of the data patterns, it also indicates that the model may be underfitting. Given that this is a linear model, underfitting could occur if the data contains non-linear relationships that the model cannot capture. Further optimization, feature engineering, or exploring non-linear extensions could help improve the model's performance.

4.4. Pegasos SVM and Logistic Regression with Polynomial Features

To enhance the capacity of the linear models to capture more complex, non-linear relationships within the data, polynomial feature expansion was applied. This technique transforms the original features into a higher-dimensional space, allowing the models to fit more intricate decision boundaries.

4.4.1. Polynomial Feature Expansion:

The original dataset, which contained 10 features, was expanded to 65 features using polynomial feature expansion of degree 2. This expansion included all pairwise interactions between features, as well as squared terms, without adding a bias term. The increased feature set provided the models with the ability to capture non-linear interactions between variables that a linear model would otherwise miss.

4.4.1.1.Model Performance:

- Pegasos SVM with Polynomial Features:
    - After re-training the Pegasos SVM with the expanded feature set, the model achieved an accuracy of 93.47%. This significant improvement over the earlier linear version indicates that the inclusion of polynomial features enabled the SVM to better capture the underlying patterns in the data, reducing the likelihood of underfitting.
- Pegasos Logistic Regression with Polynomial Features:
    - Similarly, the Pegasos Logistic Regression model, when trained with the polynomial features, achieved an accuracy of 92.81%. This marked improvement suggests that the polynomial features allowed the logistic regression model to approximate the decision boundary more accurately, leading to better classification performance.

The dramatic increase in accuracy for both models—SVM and Logistic Regression—after incorporating polynomial features highlights the importance of feature engineering in machine learning. By transforming the original linear models into non-linear ones, these models were able to fit more complex patterns in the data, effectively addressing the issue of underfitting observed with the purely linear models. The success of this approach demonstrates that feature expansion can be a powerful tool in improving model performance, especially when the relationships in the data are not strictly linear.

4.4.2. Analysis of Feature Importance in Pegasos SVM and Logistic Regression with Polynomial Features

After training the Pegasos SVM and Pegasos Logistic Regression models with polynomial features, Ianalyzed the learned weights to determine which features contributed most to the models' decision-making processes. The tables below display the top 10 features, sorted by their weight values, indicating their relative importance in each model.

Pegasos SVM Weights with Polynomial Features	
Feature	Weight
x2 * x9	2.93615
x8	1.46975
x1 * x8	1.109915
x5	0.836337
x9	0.555398
x7	0.523508
x2^2	0.379633
x1	0.37545
x1 * x2	0.281112
x5 * x8	0.209631

Pegasos Log. Regr. Weights with Polynomial Features	
Feature	Weight
x2 * x9	2.964807
x8	1.560458
x1 * x8	0.969027
x5	0.82247
x1	0.583217
x7	0.545719
x9	0.51691
x9^2	0.43267
x2^2	0.407657
x1 * x2	0.359597

In both models, the interaction term "$x_2 * x_9$" emerged as the most influential feature, with the highest weight in both the Pegasos SVM and Pegasos Logistic Regression models. This underscores the importance of capturing feature interactions when dealing with complex data. Features such as "x_8" and "$x_1 * x_8$" also appear prominently in both models, indicating their strong individual and combined influence on the classification task. The presence of these features in both tables suggests that certain patterns are consistently critical for decision-making across different linear models.

The analysis of feature weights in both the Pegasos SVM and Logistic Regression models highlights the impact of polynomial feature expansion. The significant weights assigned to interaction terms and polynomial features indicate that these models are leveraging complex, non-linear relationships to enhance their predictive accuracy. The consistency in feature importance across both models also suggests that these features are indeed crucial for effective classification in this dataset. This examination of feature importance provides valuable insights into the underlying structure of the data.

4.5. Kernelized Perceptron with RBF Kernel

The Kernelized Perceptron extends the capabilities of the classic Perceptron by incorporating a kernel function, allowing it to handle non-linear decision boundaries. This approach is particularly useful when the data is not linearly separable in its original feature space. In this implementation, Iused the Radial Basis Function (RBF) kernel, which is a popular choice for capturing non-linear patterns in data.

4.5.1. Model Implementation:

The Kernelized Perceptron was initialized with a learning rate of 1.0 and set to iterate up to 100 times over the training data. The RBF kernel, parameterized by gamma, was used to compute the similarity between data points in the feature space. The gamma parameter controls the influence of individual training examples, with smaller values leading to a broader influence and larger values making the influence more localized.

After training, the model was used to make predictions on a test set. The prediction process involved calculating the weighted sum of the similarities between the test sample and all training samples, combined with the bias, and then determining the class label based on the sign of this sum.

4.5.2. Model Performance:

The Kernelized Perceptron with the RBF kernel achieved an accuracy of **93%** on the test data. This high accuracy indicates that the model effectively captured the non-linear relationships within the data, resulting in a strong classification performance. The use of the RBF kernel allowed the Perceptron to form a non-linear decision boundary, which was crucial for handling the complexity of the dataset.

4.6. Hyperparameter Tuning for Kernelized Perceptron with RBF Kernel

In this section, I conducted hyperparameter tuning for the Kernelized Perceptron model using an RBF kernel to determine the optimal configuration for achieving the highest accuracy. The tuning process involved exploring various combinations of subset size, learning rate (learning_rate), maximum iterations (max_iter), and the RBF kernel parameter (gamma).

4.6.1. Model Implementation:

I implemented a grid search approach to evaluate the model performance across different hyperparameter settings:

- Subset Size: Two different subset sizes: 500, 1000 samples.
- Learning Rate: Learning rates of 0.1 and 0.05.
- Maximum Iterations: The number of iterations are 100, 250, and 500.
- Gamma: The gamma parameter for the RBF kernel values are 0.01 and 0.1.

For each combination of these hyperparameters, the model was trained on a subset of the data, and its accuracy was evaluated on a test set derived from that subset. The results were recorded, and the best-performing hyperparameter set was identified.

4.6.2. Analysis:

The results of the hyperparameter tuning process highlight several important insights:

- Impact of Subset Size: Increasing the subset size generally improved accuracy, up to a certain point. The subset size of 1000 provided the best balance between data availability and computational efficiency, resulting in the highest accuracy.
- Gamma Sensitivity: The gamma parameter played a crucial role in model performance. A gamma of 0.1 consistently outperformed 0.01 across different subset sizes and learning rates, indicating that a higher influence of individual data points was beneficial for capturing the underlying data structure.
- Learning Rate and Iterations: The learning rate of 0.1, combined with 250 iterations, proved optimal, allowing the model to converge effectively without overfitting.

This comprehensive tuning process demonstrates the importance of carefully selecting and optimizing hyperparameters to maximize model performance.

The best parameters were found to be a subset size of 1000, a learning rate of 0.1, 250 iterations, and a gamma of 0.1, achieving the highest accuracy of 96.50%.

4.6.3. Full Dataset Evaluation:

After conducting initial evaluations on a smaller subset of the data, the Kernelized Perceptron with an RBF kernel was trained on the full training dataset and subsequently evaluated on the test dataset. This approach allows us to assess how well the model generalizes to unseen data after being trained on all available training data.

The Kernelized Perceptron was trained using the entire training dataset (X_train, y_train), with the model configured to run for 1000 iterations. This comprehensive training ensures that the model has access to all the patterns and relationships present in the training data. After training, the model was tested on the test dataset (X_test, y_test). The model achieved an accuracy of 95.50% on the test data. This high accuracy suggests that the Kernelized Perceptron with the RBF kernel is well-suited for this classification task, capturing the underlying patterns effectively.

4.7. Kernelized Pegasos SVM with RBF and Polynomial Kernels

The Kernelized Pegasos SVM extends the linear Pegasos algorithm by incorporating kernel functions, enabling it to handle non-linear decision boundaries. In this experiment, Iimplemented and evaluated the Kernelized Pegasos SVM using both the Radial Basis Function (RBF) kernel and the Polynomial kernel. The goal was to determine the best-performing kernel and to optimize the model's hyperparameters for maximum accuracy.

4.7.1. Model Implementation:

The Kernelized Pegasos SVM was implemented with support for both RBF and Polynomial kernels. The kernel functions were defined as follows:

- RBF Kernel**:** Measures the similarity between two samples using a Gaussian function, controlled by the gamma parameter, which determines the width of the Gaussian.
- Polynomial Kernel: Computes similarity using a polynomial function with parameters degree and coef0

The model was trained using the Pegasos algorithm, where the alphas were updated iteratively based on whether the current sample violated the margin defined by the SVM objective function. The training process was repeated for a maximum number of iterations (max_iter), with the learning rate (eta) decreasing over time.

Iperformed hyperparameter tuning to identify the optimal configuration for the RBF kernel. The following hyperparameters were tuned:

- lambda_param: The regularization parameter, controlling the trade-off between maximizing the margin and minimizing classification error.
- gamma: The parameter for the RBF kernel, controlling the influence of individual training samples.
- max_iter: The maximum number of iterations for the Pegasos algorithm.

4.7.2. Analysis:

The Kernelized Pegasos SVM with the RBF kernel did not perform as well as expected, achieving an accuracy of 48.21% on the test data. This result highlights the importance of carefully tuning the hyperparameters and possibly experimenting with different kernel functions to better capture the data's structure.

5. CONCLUSION

The data preprocessing phase involved meticulous steps, including feature scaling using StandardScaler and RobustScaler and outlier removal with the Interquartile Range (IQR) method. These preprocessing efforts resulted in a well-prepared dataset, with features exhibiting symmetrical distributions and reduced skewness, essential for the optimal performance of machine learning models.

Model	Feature Set	Accuracy (%)
Kernelized Perceptron (RBF Kernel)	Full Dataset (Tuned Hyperparameters)	95.50%
Pegasos SVM	Polynomial Features (Degree 2)	93.47%
Kernelized Perceptron (RBF Kernel)	Original Features (Subset)	93.00%
Pegasos Logistic Regression	Polynomial Features (Degree 2)	92.81%
Pegasos SVM	Original Features	73.17%
Pegasos Logistic Regression	Original Features	70.40%
Perceptron	Original Features	67.61%
Kernelized Pegasos SVM (RBF Kernel)	Original Features	48.21%

Initially, three fundamental models were implemented: the Perceptron, Pegasos SVM, and Pegasos Logistic Regression. The Perceptron achieved a modest accuracy of 67.61%, indicating potential underfitting due to its linear nature. The Pegasos SVM and Logistic Regression models improved upon this, with accuracies of 73.17% and 70.40%, respectively, suggesting better pattern recognition but still limited by their linear frameworks.

Polynomial feature expansion leading to significant performance improvements. The Pegasos SVM and Logistic Regression with polynomial features achieved accuracies of 93.47% and 92.81%, respectively, highlighting the effectiveness of feature engineering in capturing complex data patterns. Furthermore, the Kernelized Perceptron with an RBF kernel achieved an accuracy of 95.50% after hyperparameter tuning, demonstrating its ability to model non-linear decision boundaries effectively.

However, an attempt to extend the Pegasos SVM with kernel methods resulted in a lower accuracy of 48.21%, underscoring the importance of careful kernel selection and hyperparameter tuning in such models.

In conclusion, this study underscores the critical role of data preprocessing, feature engineering, and model selection in achieving high performance in machine learning tasks. While simple linear models may provide a baseline, they are often insufficient for datasets with complex, non-linear relationships. The significant improvements observed with polynomial feature expansion and kernel methods highlight the importance of transforming the feature space to capture intricate patterns in the data. The Kernelized Perceptron with an RBF kernel emerged as the best-performing model, demonstrating that combining kernel methods with traditional algorithms is highly effective for handling non-linearities in data.