UNIVERSITÀ DEGLI STUDI DI MILANO

Kernelized Linear Classification

AUTHOR

Michele Coaro - 41208A

Abstract

This report details the analysis performed through the creation and implementation of different linear classification approaches, recreating core algorithms both in their standard and kernelized forms. Specifically, the study focuses on the Perceptron, Pegasos for Support Vector Machines (SVMs), and Regularized Logistic Regression. For Perceptron and Pegasos, kernelized extensions have been introduced to capture more complex decision boundaries. Additionally, second-degree polynomial feature expansion was adopted to enhance model flexibility, thereby aiming to improve the performance of traditionally linear methods. The primary objective was to predict the target label y according to the $0-1$ loss metric. Key components of this work included thorough data preprocessing, measures against data leakage, and hyperparameter tuning to maximize each algorithm's accuracy.

From the experimental outcomes, both polynomial expansion and kernelized techniques substantially boosted classification accuracy, highlighting their effectiveness over purely linear approaches.

Contents

1 Introduction

Kernelized linear classification is meant to improve the capabilities of traditional linear models by incorporating kernel functions to capture non-linear relationships, thus uncovering underlying dynamics and higher order interactions between variables. In this work, implementations of different algorithms, such as the Perceptron, the Pegasos for Support Vector Machines, and Regularized Logistic Classification (or Regression) were developed in Python from scratch. These models were further enhanced with kernel techniques, specifically Gaussian and polynomial kernels, in an attempt at improving their predictive performance. Additionally, second-degree polynomial feature expansion was employed to introduce non-linear transformations of the original data.

The remainder of this report is organized as follows:

- **Section 2** describes dataset exploration and preprocessing procedures, highlighting consistency checks and measures taken to avoid data leakage.
- **Section 3** outlines the methodology, including algorithm details, tuning of hyperparameters, and considerations for model implementation.
- **Section 4** presents the results of the various techniques, emphasizing the impact of kernel methods and polynomial expansion on performance.
- **Section 5** concludes the report, summarizing key findings and reflecting on potential directions for future analysis.

The overall objective was to develop a sound methodology that addresses possible complexities such as data handling, hyperparameter tuning, and efficient algorithm design. All analyses were performed in two components: a Python file (`ffunctions.py`) containing the core functions, and a Jupyter notebook (`fnotebook.ipynb`) demonstrating and visualizing the results. The prefix “f” in the file names playfully commemorates, standing for "finally", the lengthy effort invested in this project.

2 Dataset Exploration and Preprocessing

The dataset under consideration contains ten numerical variables, denoted by $x_1, \ldots, x_{10}$. An exploratory data analysis (EDA) was performed to understand the distribution and structure of these features. The following steps summarize the preprocessing procedures that ensured consistent data handling:

1. **Data Cleaning:** Missing values and outliers were identified and handled appropriately.
2. **Normalization/Standardization:** Feature scaling was applied to ensure numerical stability during model training.
3. **Multicollinearity:** Highly correlated features were identified and removed to prevent redundancy.
4. **Data Splitting:** The dataset was partitioned into training and test sets, ensuring any scaling procedures were fit on training data only, thereby avoiding data leakage.

Figure 1 illustrates the distribution of key features before and after normalization. Initial checks revealed a small number of outliers, which were treated as noise based on their low incidence (fewer than eighty instances in a dataset of roughly ten thousand observations). These points were removed using a z-score threshold. Following this cleaning, the dataset was divided into training and test sets so that each model could be evaluated on unseen data.

To further reduce the possibility of data leakage, statistical parameters for normalization (mean and standard deviation) were computed exclusively from the training set and then applied to both training and test partitions. A correlation matrix (Figure 2) highlighted a strong association among three features: x_3, x_6, and x_{10}. Consequently, two of these columns were removed to avoid redundancy and reduce the risk of distorted model performance. The decision on which column to remove has not been included in the report, as it is a tedious process consisting on running the same algorithms on different versions of the dataset each missing one or two of the involved variables.

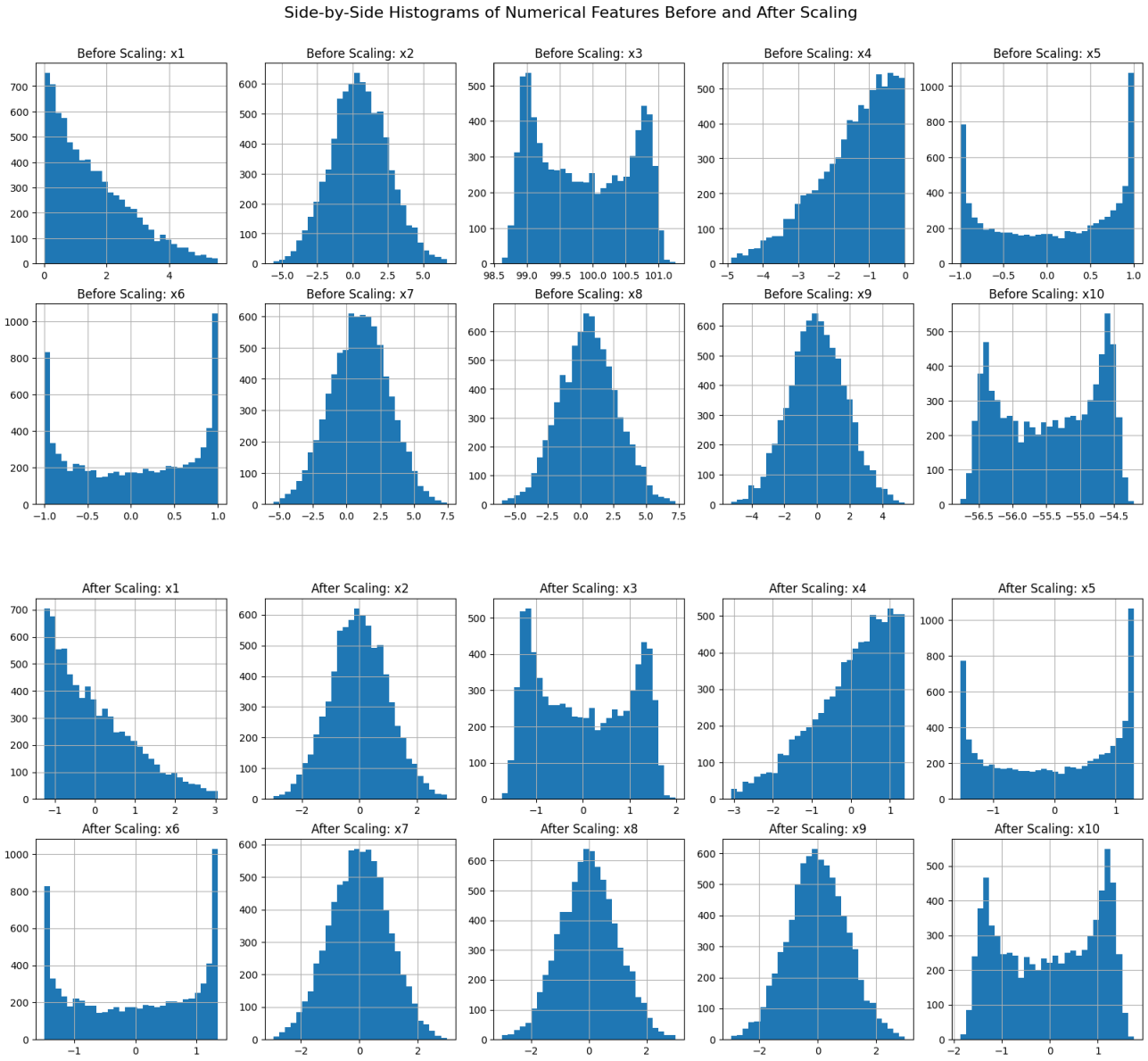

Figure 1: Numerical features before and after scaling

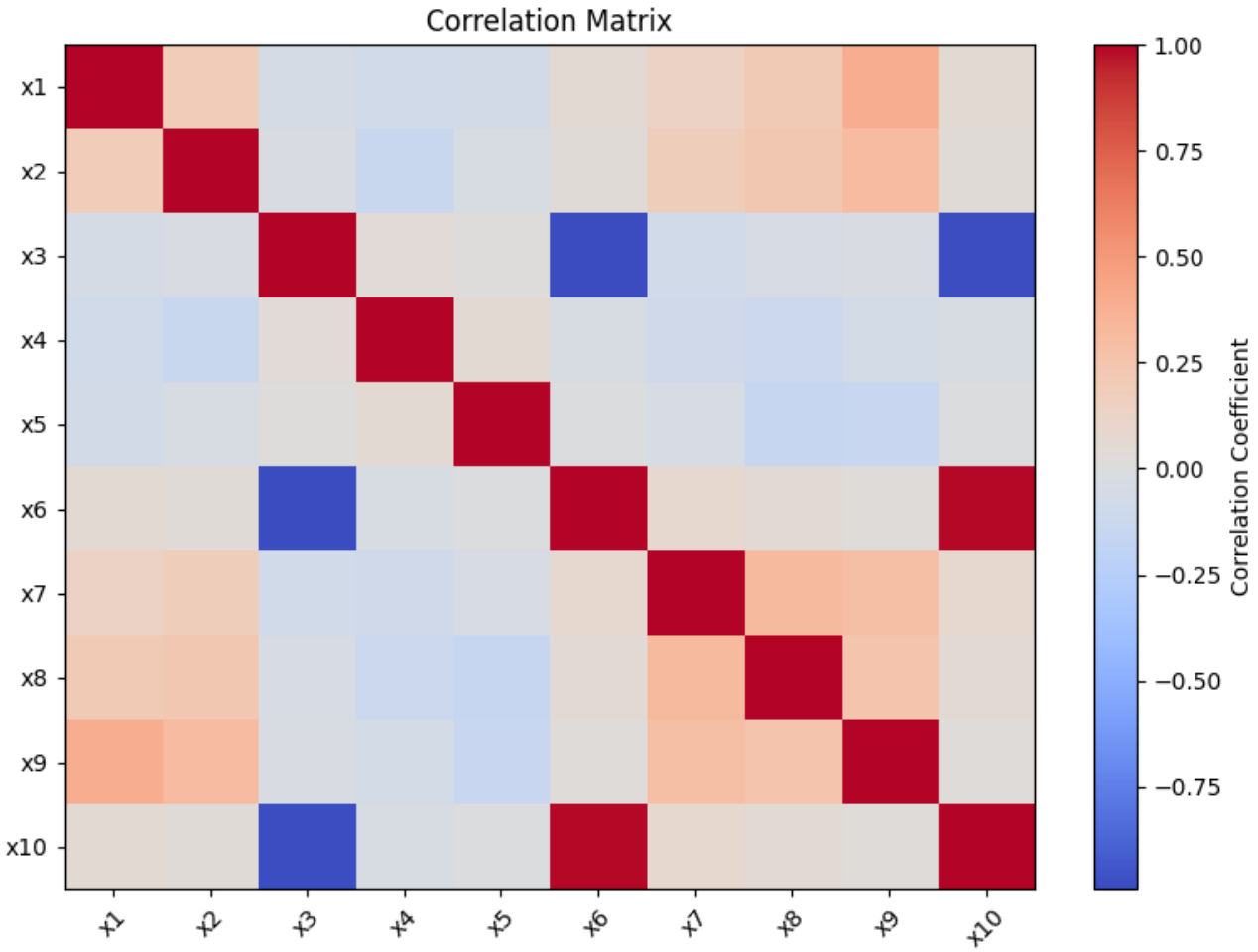

Figure 2: Correlation matrix for the ten numerical features

3 Methodology

This section provides an overview of the classification algorithms, polynomial feature expansion, and kernel methods used in this study. It also describes the hyperparameter tuning strategies applied to optimize model performance across different experiments.

3.1 Base Classification Algorithms

Three primary classification methods were implemented from scratch:

3.1.1 Perceptron Algorithm

The Perceptron is an iterative method for binary classification. Given a training sample $\big(x_i, y_i\big)$ where $y_i \in \{-1, +1\}$, the algorithm updates its weight vector w when a misclassification occurs:

$$w \leftarrow w + \eta\, y_i\, x_i,$$

where η is the learning rate. This rule shifts the decision boundary based on misclassified examples to reduce classification errors over time.

3.1.2 Pegasos Algorithm for SVM

Pegasos is a stochastic sub-gradient descent approach for solving the SVM optimization problem:

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

where λ is the regularization parameter, and n is the number of training examples. The algorithm alternates between sub-gradient updates and weight projection, balancing the hinge loss and the regularization term.

3.1.3 Regularized Logistic Classification

Regularized Logistic Classification (or Regression), serves as a probabilistic approach to binary classification. Its objective function is:

$$\min_{w} \quad \frac{1}{n}\sum_{i=1}^{n} \log\big(1 + \exp(-y_i\, w^\top x_i)\big) \;+\; \frac{\lambda}{2}\,\|w\|^2,$$

which includes an L_2 regularization term to mitigate overfitting by constraining the magnitude of w.

3.2 Polynomial Feature Expansion

To allow linear models to capture non-linear relationships, a second-degree polynomial feature expansion was applied. This transformation maps an original feature vector $x = [\, x_1,\, x_2,\, \ldots,\, x_d \,]$ to an expanded vector $\phi(x)$ that includes squared terms ($x_1^2, x_2^2, \ldots$) and pairwise interactions ($x_1x_2, x_1x_3, \ldots$). By incorporating these higher-order terms, models can learn more complex decision boundaries without explicitly shifting to a non-linear algorithm.

3.3 Kernelized Methods

Kernel methods were introduced to further enhance the ability of Perceptron and Pegasos to capture non-linear patterns. Instead of computing feature expansions explicitly, these approaches rely on kernel functions to implicitly operate in a higher-dimensional space.

3.3.1 Kernelized Perceptron

A kernelized Perceptron was implemented using two kernel functions:

- **Gaussian Kernel:**
$$K(x, x') \;=\; \exp\!\Big(-\tfrac{\|x-x'\|^2}{2\,\sigma^2}\Big),$$
where σ controls the kernel width.
- **Polynomial Kernel:**
$$K(x, x') \;=\; \big(\alpha\,\langle x, x'\rangle \;+\; c\big)^d,$$
where α, c, and d (chosen as 2 for these experiments) parameterize the polynomial expansion.

3.3.2 Kernelized Pegasos SVM

A kernelized variant of the Pegasos SVM was also implemented with the same Gaussian and polynomial kernels. By replacing inner products in the Pegasos update with kernel evaluations, the model learns non-linear decision boundaries without explicitly enumerating higher-order features.

3.4 Hyperparameter Tuning

A systematic hyperparameter tuning process was conducted to identify optimal settings for each model. Cross-validation and grid search were used to adjust parameters such as:

- **Learning Rate (η):** Governs the update step size for Perceptron and Pegasos.
- **Regularization Parameter (λ):** Determines the trade-off between model complexity and fitting accuracy.

Table 1 summarizes the best hyperparameters observed through cross-validation, along with corresponding accuracy values. For the linear Perceptron, `epochs` $= 10$ and `eta` $= 0.01$ achieved an accuracy of 0.6708. Pegasos SVM performed best with `lambda` $= 0.1$ and `epochs` $= 20$, reaching 0.7314. Regularized Logistic Regression attained 0.7239 under `lambda` $= 0.1$, `epochs` $= 10$, and `eta` $= 0.1$.

Table 1: Best hyperparameters and cross-validation accuracies. Dashes (-) indicate parameters not applicable.

Method	**epochs**	**eta**	**cv_accuracy**	**lambda**	**sigma**
Perceptron	10.0	0.01	0.670817	—	—
Pegasos	20.0	—	0.731388	0.100	—
Logistic	10.0	0.10	0.723865	0.100	—
PerceptronPoly	10.0	0.01	0.928664	—	—
PegasosPoly	20.0	—	0.944099	0.001	—
LogisticPoly	10.0	0.10	0.893256	0.100	—
KernelPerceptronGauss	5.0	—	0.923476	—	1.0

Feature expansion to second-degree polynomials significantly boosted performance for both Perceptron (`PerceptronPoly`) and Pegasos (`PegasosPoly`). Meanwhile, kernel-based variants, especially those employing Gaussian kernels, also showed competitive results. These outcomes underscore the importance of carefully tuning hyperparameters for each algorithm and variant to harness their full potential.

4 Results

The code implementation facilitates clear comparisons of the performance across different models. The primary evaluation metric is *accuracy*, defined as the proportion of correctly classified instances within the test set:

$$\text{Accuracy} \;=\; \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\{\hat{y}_i = y_i\},$$

where n is the total number of test samples, $\hat{y}_i$ is the predicted label for the i-th sample, and y_i is the true label. This metric corresponds to minimizing the 0–1 loss,

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

since

$$\text{Accuracy} \;=\; 1 - L_{0-1}.$$

Figure 3 illustrates an aggregated view of the results.

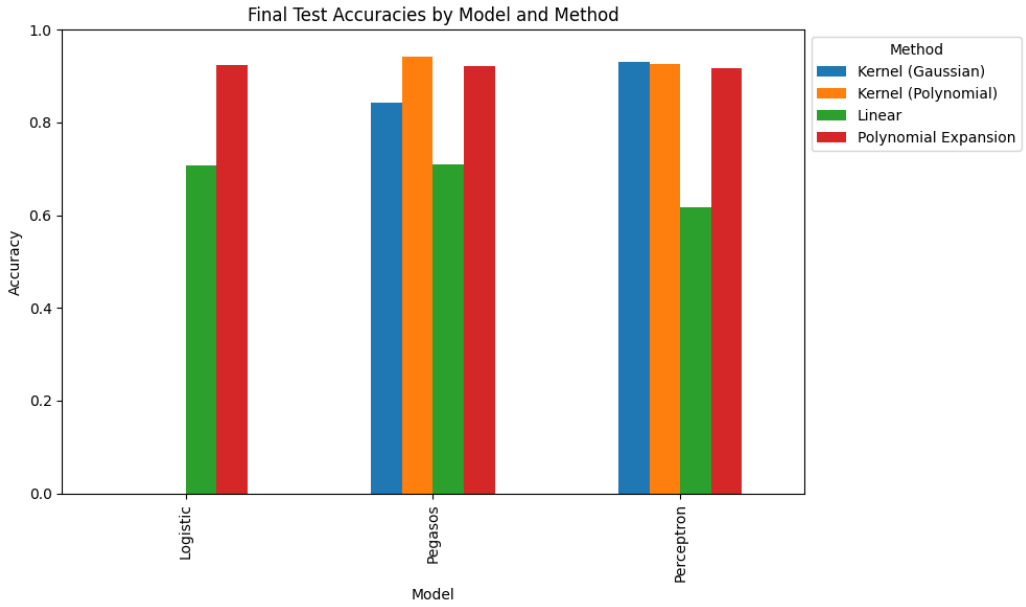

Figure 3: Results grouped by method.

4.1 Perceptron

The Perceptron algorithm is a simple yet effective linear classifier that updates its weight vector whenever a training instance is misclassified. Four variants were considered:

- **Linear Perceptron**: Uses the original input features without modification.
- **Perceptron with Polynomial Feature Expansion**: Incorporates a degree-two polynomial expansion to capture quadratic relationships.
- **Kernelized Perceptron (Gaussian Kernel)**: Operates in a higher-dimensional space via a Gaussian kernel.
- **Kernelized Perceptron (Polynomial Kernel)**: Employs a degree-two polynomial kernel, implicitly modeling the effect of feature expansion.

The following pseudocode outlines the standard (linear) Perceptron:

```
Algorithm: Perceptron
Input: training data {(x_i, y_i)}, learning rate Î·, number of
   epochs T
Initialize: w = 0
For t = 1 to T:
    For each (x_i, y_i) in training data:
        If y_i * (w^T * x_i) <= 0 then:
            w = w + Î· * y_i * x_i
Return w
```

In the kernelized version, the dot product $w^\top x_i$ is replaced by a kernel evaluation $K(x_i, x_j)$, where K is either the Gaussian or the polynomial kernel:

$$\text{Gaussian kernel: } K(x, x') = \exp\!\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right), \quad \text{Polynomial kernel: } K(x, x') = (\alpha \langle x, x' \rangle + c)^2.$$

Experimental Findings: The linear Perceptron yielded an accuracy of $\mathbf{0.617360}$. Introducing a second-degree polynomial expansion increased accuracy to $\mathbf{0.917309}$. Further improvements were observed with kernelization: the Gaussian kernel achieved $\mathbf{0.930149}$ and the polynomial kernel reached $\mathbf{0.926554}$. These results underscore the substantial benefits of non-linear transformations for Perceptron-based methods.

4.2 Pegasos SVM

Pegasos is a stochastic sub-gradient descent approach to training Support Vector Machines. Four variations were implemented:

- **Linear Pegasos SVM**: Applies Pegasos updates on the original features.
- **Pegasos SVM with Polynomial Feature Expansion**: Explicitly expands features to degree two.
- **Kernelized Pegasos (Gaussian Kernel)**: Uses a Gaussian kernel in place of direct inner products.
- **Kernelized Pegasos (Polynomial Kernel)**: Employs a polynomial kernel for implicit feature mapping.

The pseudocode for linear Pegasos is shown below:

```
Algorithm: Pegasos SVM
Input: training data {(x_i, y_i)}, regularization parameter Î»,
       learning rate Î·, number of iterations T
Initialize: w = 0
For t = 1 to T:
    Randomly select (x_i, y_i) from training data
    If y_i * (w^T * x_i) < 1 then:
        w = (1 - Î·Î») * w + Î· * y_i * x_i
    Else:
        w = (1 - Î·Î») * w
Return w
```

In kernelized versions, the inner product $w^\top x_i$ is replaced by the corresponding kernel function.

Experimental Findings: Linear Pegasos SVM attained an accuracy of $\mathbf{0.709296}$. Polynomial expansion raised that accuracy to $\mathbf{0.920904}$. When kernel methods were applied, the Gaussian kernel achieved $\mathbf{0.842322}$, while the polynomial kernel obtained $\mathbf{0.941448}$. These outcomes show that nonlinear transformations, particularly via polynomial kernels, can substantially enhance Pegasos-based classification.

4.3 Regularized Logistic Classification

Regularized Logistic Regression is a probabilistic classifier incorporating an L_2 penalty to reduce overfitting. The linear version can be summarized in gradient descent form:

```
Algorithm: Regularized Logistic Regression
Input: training data {(x_i, y_i)}, learning rate Î·,
       regularization parameter Î», number of iterations T
Initialize: w = 0
For t = 1 to T:
    Compute gradient:
      g = (1/n) * Î£ [ -y_i * x_i * sigmoid(-y_i * (w^T * x_i)) ]
          + Î» * w
    Update: w = w - Î· * g
Return w
```

Two versions were examined:

- **Linear Logistic Regression**: Used without feature transformations.
- **Logistic Regression with Polynomial Feature Expansion**: Leveraged a degree-two polynomial expansion.

Experimental Findings: The linear logistic classifier delivered an accuracy of $\mathbf{0.708269}$. Applying a polynomial expansion significantly increased the accuracy to $\mathbf{0.925013}$. No kernel-based experiments were conducted for logistic regression in this analysis.

4.4 Summary of Results

Table 2 and Figure 4 provide an overview of the accuracies obtained for each model and its corresponding variant:

Model	Kernel (Gaussian)	Kernel (Polynomial)	Linear	Poly. Expansion
Logistic	—	—	0.708269	0.925013
Pegasos SVM	0.842322	0.941448	0.709296	0.920904
Perceptron	0.930149	0.926554	0.617360	0.917309

Table 2: Accuracies for each model variant. Kernel methods were not applied to Logistic Regression.

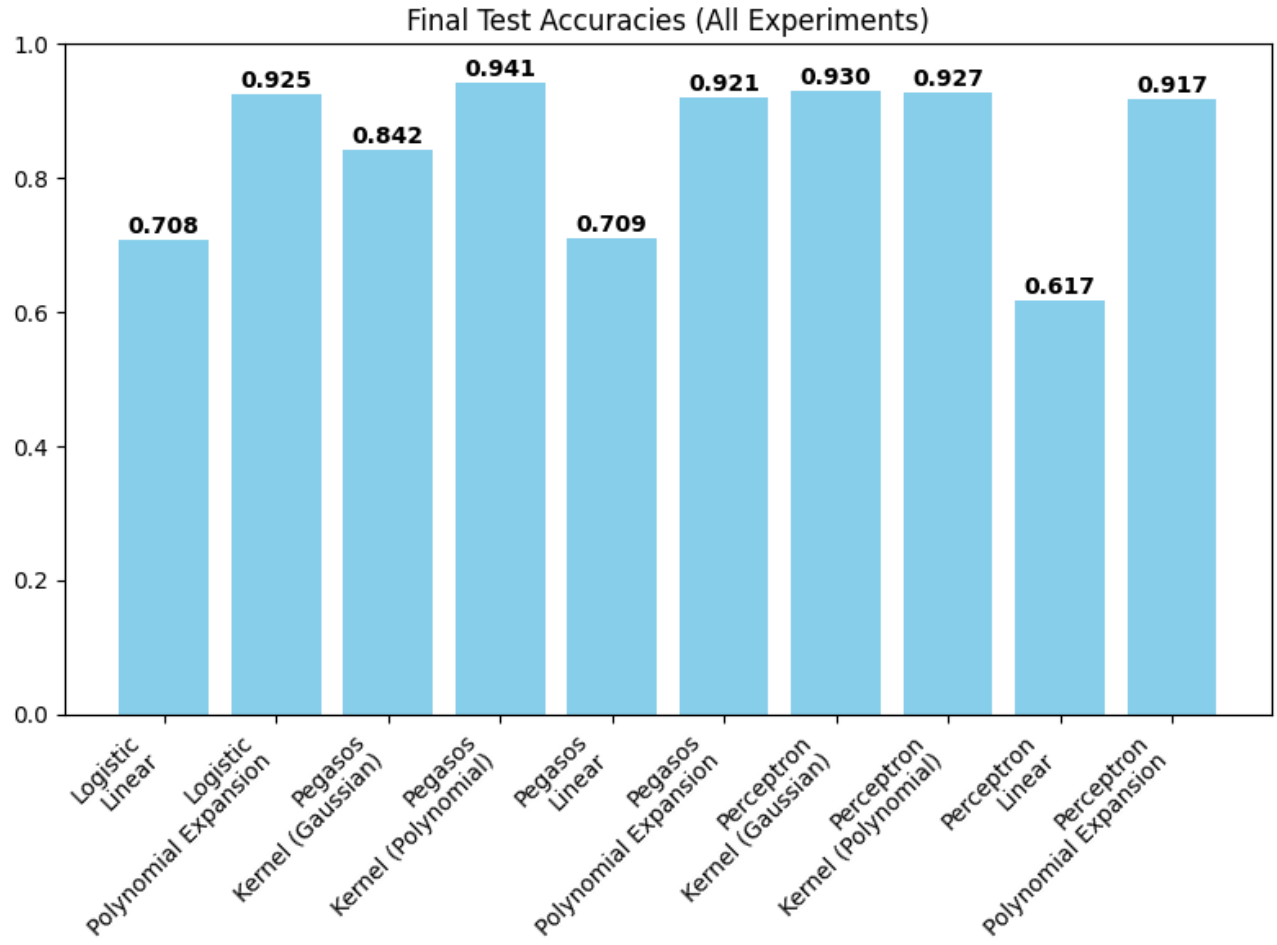

Figure 4: Final result on test set comparison across all models

4.5 Assessment of under/overfitting

An important aspect of model evaluation involves comparing cross-validation performance to results on the final test set. In this study, the cross-validation accuracies across different models and parameter settings align closely with the outcomes observed on the test set. Such consistency suggests that the classifiers are neither significantly overfitting (i.e., memorizing the training data) nor underfitting (i.e., failing to capture relevant patterns). No critical evidence of overfitting or underfitting has been discovered, thus signaling the reliability of the training and validation procedures employed.

These findings confirm that introducing non-linear transformations, either through polynomial feature expansion or kernel functions, leads to higher accuracy scores and consequently a lower 0–1 loss.

5 Conclusion

In this study, several classification methods were implemented and evaluated, including the Perceptron, Pegasos SVM, and Regularized Logistic Regression, to examine how non-linear transformations affect model performance. Polynomial feature expansion (degree two) and, when applicable, kernel methods (Gaussian and polynomial kernels) were introduced to the base models. The primary performance metric was accuracy:

$$\text{Accuracy} \;=\; \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{\hat{y}_i = y_i\},$$

which relates directly to the 0–1 loss:

$$L_{0-1} \;=\; \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{\hat{y}_i \neq y_i\} \quad \text{and} \quad \text{Accuracy} \;=\; 1 \;-\; L_{0-1}.$$

Key Findings:

- **Enhanced Performance via Non-linear Transformations:** Both the Perceptron and Pegasos SVM benefited significantly from polynomial feature expansion or kernelization. For example, the linear Perceptron achieved an accuracy of 0.617360, which increased to more than 0.917 with polynomial expansion and approximately 0.930 with kernel methods. The linear Pegasos SVM similarly rose from 0.709296 to over 0.920 when features were explicitly expanded, and to 0.941 with the polynomial kernel.
- **Logistic Regression Observations:** Although kernel methods were not investigated for Logistic Regression in this work, the regularized logistic model also realized substantial gains from polynomial expansion. Its accuracy improved from 0.708269 in the linear version to 0.925013 with second-degree features.
- **Trade-offs:** While non-linear transformations generally yielded higher accuracies, they also introduced greater model complexity and computational demands. Kernel-based methods, in particular, showed sensitivity to hyperparameter choices, underscoring the importance of thorough tuning and resource considerations in practice.

Concluding Remarks: Overall, these results demonstrate that non-linear transformations, implemented either via polynomial expansion or through kernel functions, substantially enhance the predictive power of linear classifiers. Increases in accuracy (equivalent to reductions in 0–1 loss) confirm the benefits of these techniques. However, the higher computational requirements and the need for careful hyperparameter selection indicate that performance must be weighed against computational costs.

I declare that this material, which is now submitted 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. I understand that plagiarism, collusion, and copying are serious offenses and accept the associated penalties. This assignment, or any part of it, has not been previously submitted for assessment in this or any other course of study.

List of Figures