Experimental Project 1: Kernelized Linear Classification

Akash Mittal

Università degli Studi di Milano Statale - UNIMI

B79: Data Science for Economics

Matricola ID : 31909A

Contents

Introduction

This objective of the project is to explore the different machine learning models to classify the given dataset (with 10,000 rows and 11 columns (10 features and 1 target variable) according to the labels following a zero-one loss function. The dataset contains numerical features and classification is performed according to zero-one loss with suitable data preprocessing. The models used for the project are 1) Perceptron; 2) SVM with Pegasos algorithm; 3) Regularized Logistic Regression (with Pegasos Objective Function); 4) All three (1,2,3) models with data expanded using Polynomial Expansion for “degree 2”; 5) Kernelized Perceptron with Polynomial & Gaussian Kernels; 6) Kernelized Pegasos with Gaussian & Polynomial Kernel. Cross-validation and hyperparameter tuning is performed for a robust approach.

Data Import and Processing

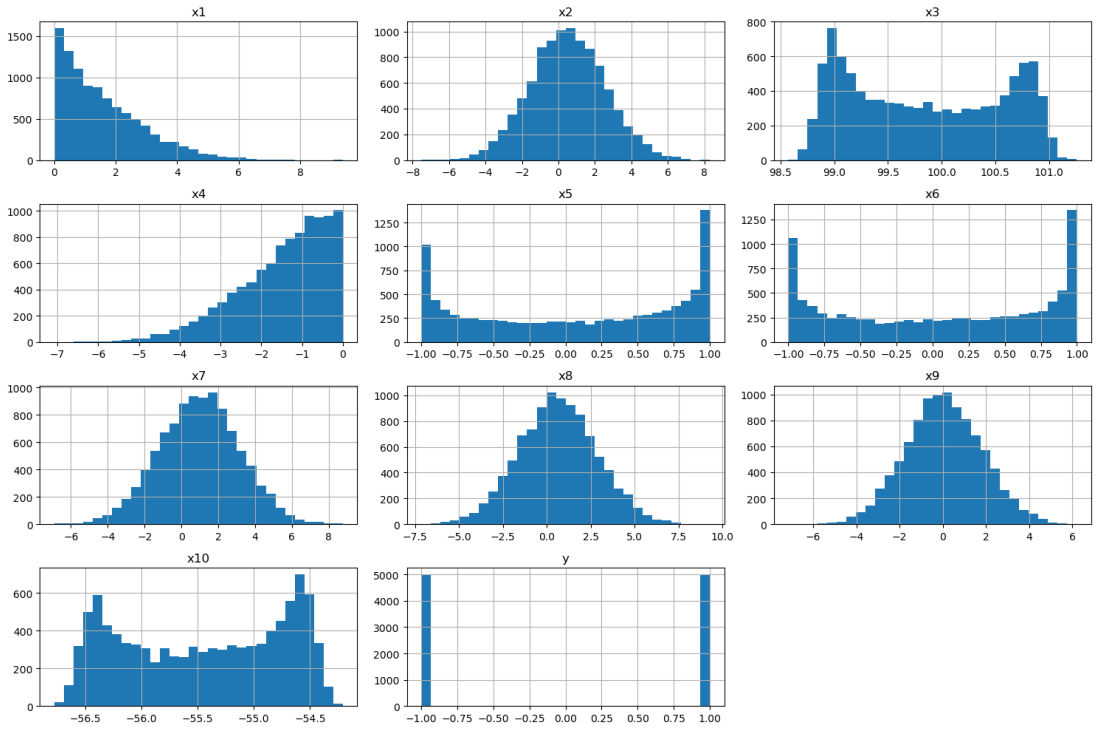
Data Loading and Exploration

The dataset for the project is given and it is a random dataset with 10 features and 1 label and dataset includes 10000 rows × 11 columns. The features are named x1:x10 and y for labels. The labels are {+1, -1}. The dataset has no null values and no duplicates.

Source of Dataset:

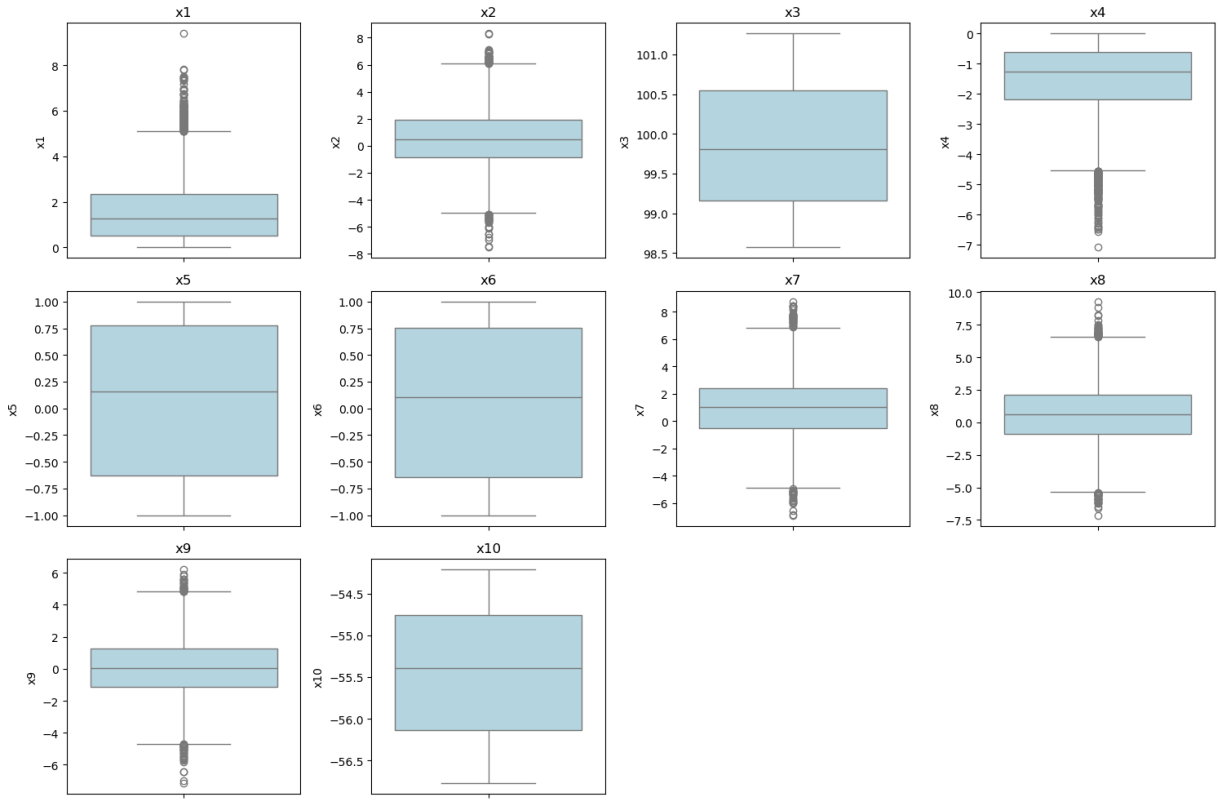
https://www.google.com/url?q=https://drive.google.com/file/d/1HeP6uwyG3oMcJnViG4yCMmzmSeTa1mko/view?usp%3Ddrive_link&sa=D&source=editors&ust=1722278116855959&usg=AOvVaw0k76s77tkTU4xrC59pzfEO

The dataset was loaded using pandas from a “CSV file”. Descriptive statistics of the dataset were examined to understand its distribution and detect any irregularities. Visualizations such as histograms and boxplots were generated for each feature to better understand their distributions and identify potential outliers. The plots are shown below:



Histogram of the features in dataset

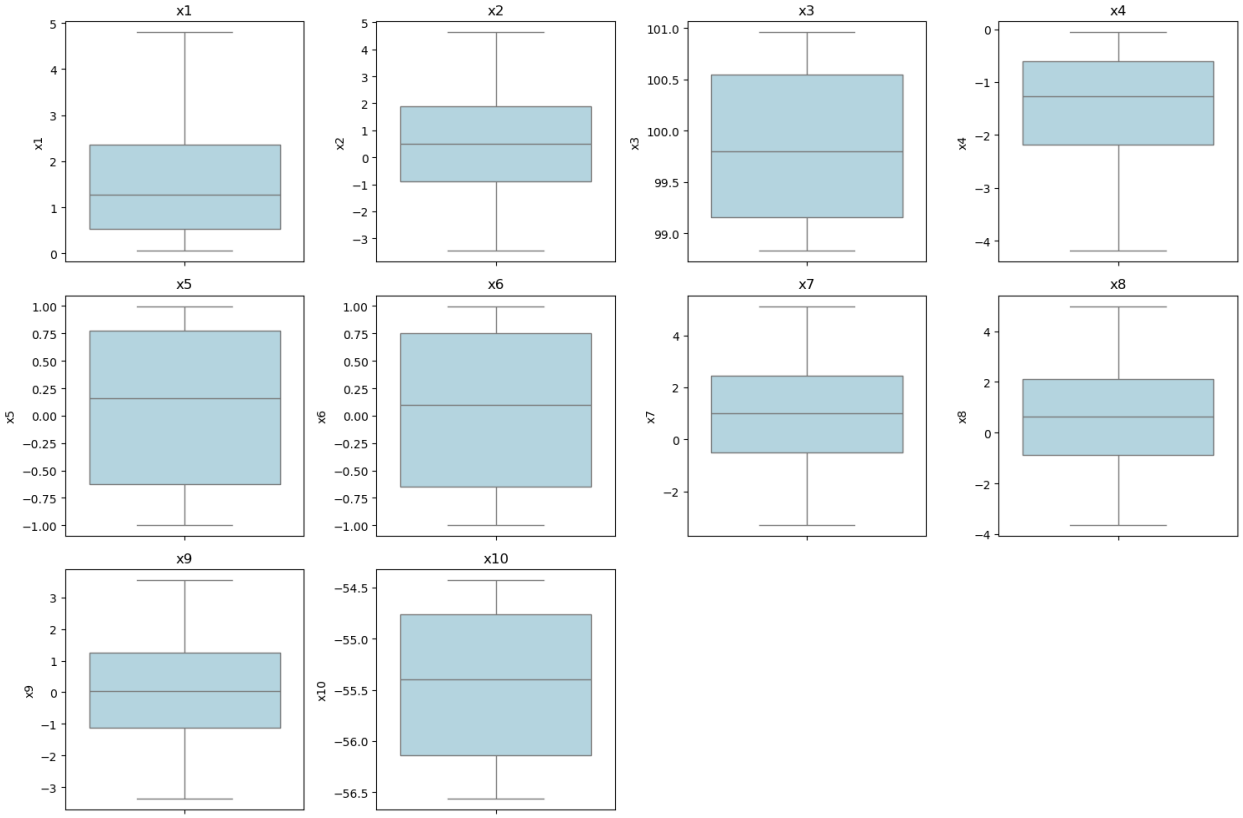
The plot of target variable shows that data is balanced in the two classes. Further, the histogram also shows that some of the features are skewed, e.g. x1, x4, and some are bimodal, e.g. x3, x5, x6 and x10.



Boxplot of the features showing presence of outliers

Data Preprocessing

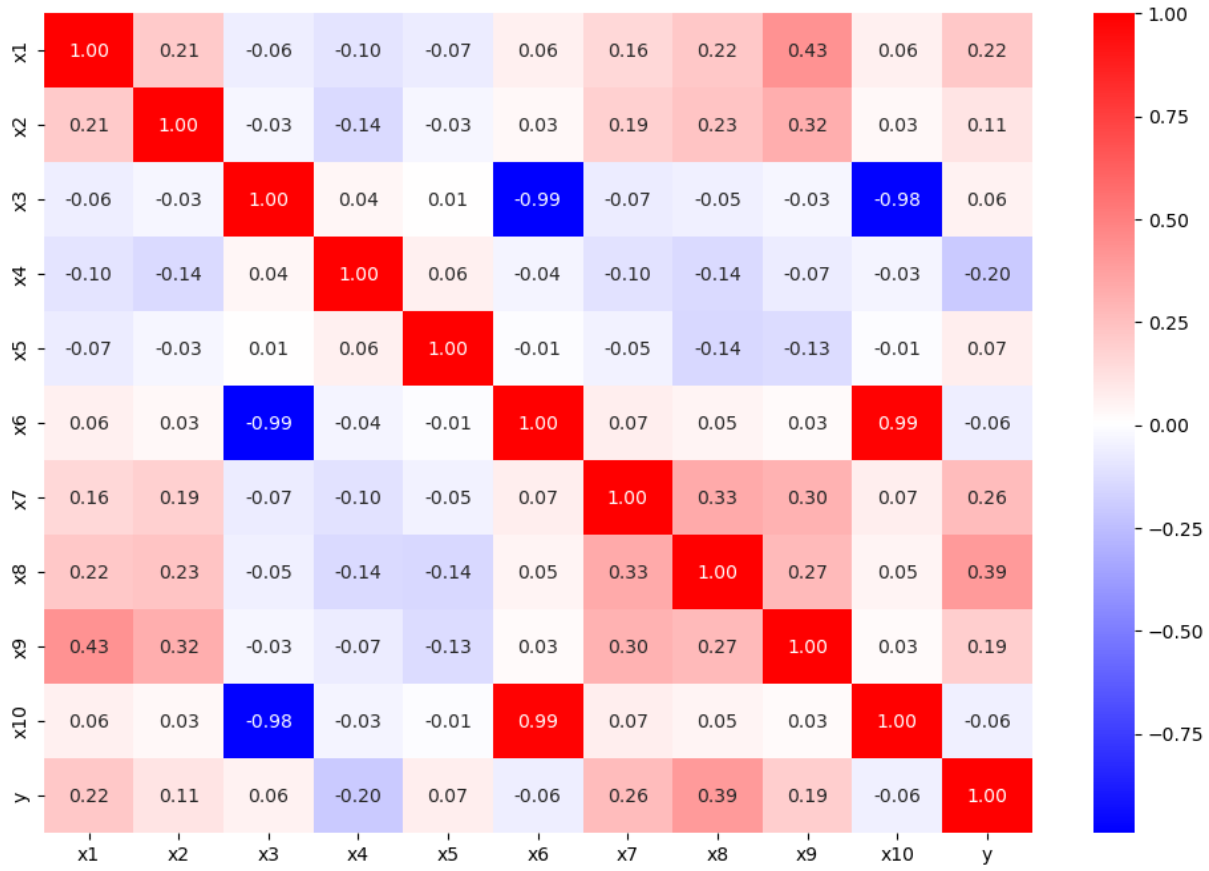
Outliers were handled using Winsorization, that limits extreme values by adjusting them to a less extreme value at a specified percentile. This was done for all features except the target variable. Further, as per the data for winsorization, a 5% level was chosen for outliers, 2.5% each on the lower and upper side of tails.



Boxplot of the features after Winsorization

Correlation Plot:

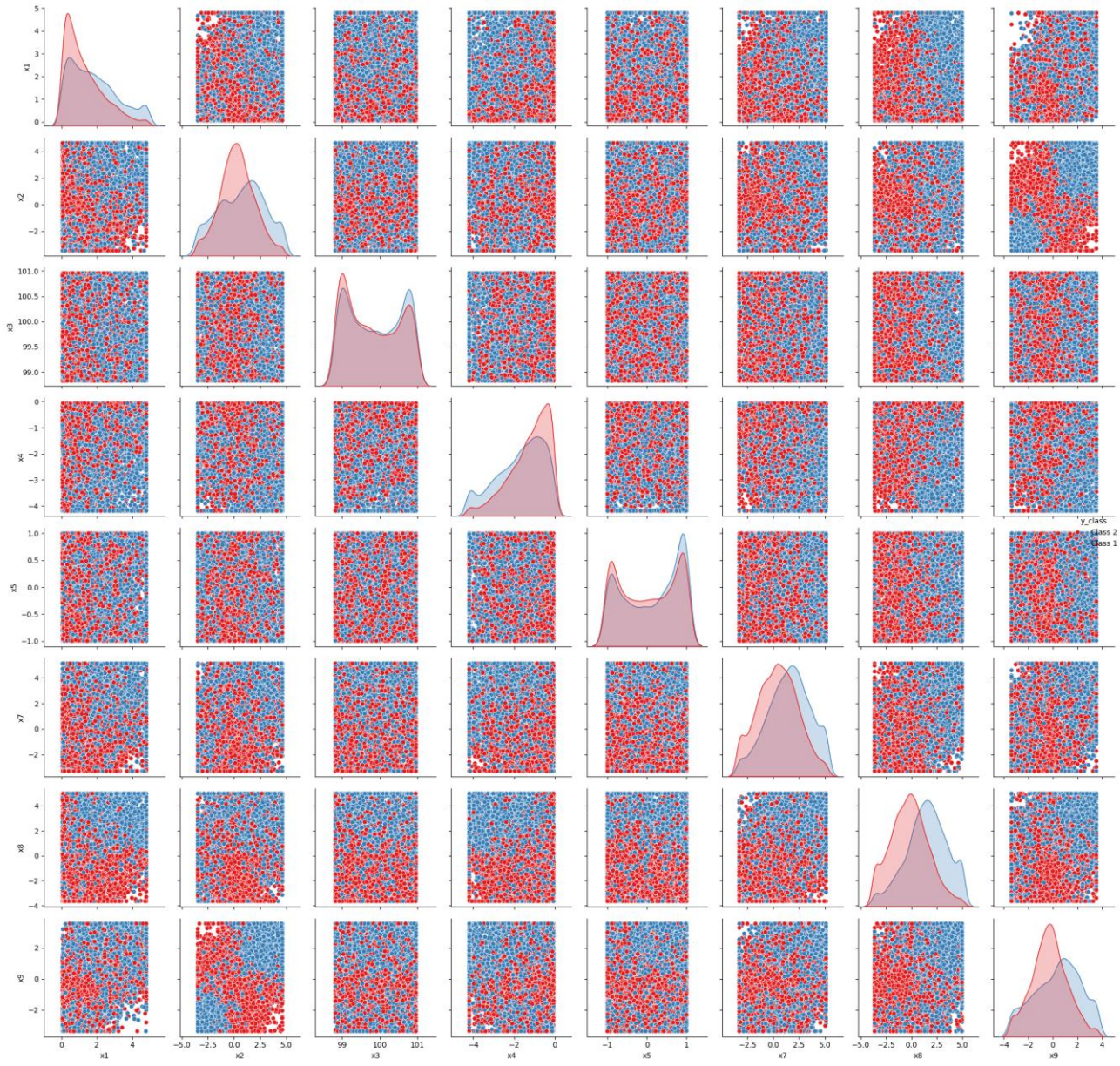
Below is the correlation plot for the features to see if there are features with very high correlation among them. The correlation plot shows a very high correlation ~0.99 for the variables x3, x6 and x10. Now, since there is no information about the description of the different variables, the variables x6 and x10 are dropped from the dataset.



Correlation Plot

Pair Plots:

After the winsorization and removal of correlated features, the dataset was visualized using pair plots to show relationships between variables, with the target variable ‘y’ being color-coded for better understanding.



Now, the below scatter plot shows the distribution of data in the two different classes as per the labels in the given dataset. It shows that data is not directly distinguishable and would need careful implementation of the algorithms to separate the data in the two classes.

Data Splitting

The dataset was split into training and testing sets using a custom “TrainTestSplit” class. Stratified splitting was used to ensure that the class distribution in both training and testing datasets remained balanced. The training set was set to 75% and the test size was set to 25%.

Feature standardization was performed on the training features to ensure that each feature in the dataset contributes equally to the model. Standardization was performed by scaling the features to centre them around mean = 0 and standard deviation = 1.

- **Fitting on Training Data**: The training data is used to compute the mean and standard deviation for each feature. These values are then used to scale both the training and test sets. Thus, avoiding data leakage, which would have happened if the test data was used in computing the scaling parameters.
- **Test Data Transformation**: The test data is then scaled using the mean and standard deviation derived from the training data to ensure that the model’s performance is evaluated in a consistent manner.

After standardization, the data is converted to array (i.e. NumPy array) for compatibility with machine learning algorithms. The conversion ensures that the data is in the appropriate numerical format for optimal performance during training and evaluation. The shape of the datasets afterwards is provided below –

X_train - (7500, 8)

X_test - (2500, 8)

y_train - (7500,)

y_test - (2500,)

Further, unique labels in the training and test sets were also cross checked (to be +1, -1) to void any mishaps.

Machine Learning Models

Perceptron Model

Perceptron is a supervised learning algorithm used for binary classification tasks. The main goal of the Perceptron algorithm is to learn a decision boundary that separates two classes of data. The Perceptron algorithm makes predictions based on a linear combination of input features, passed through an activation function, e.g. sign function which outputs either +1 or -1 depending on the sign of the input.

The Perceptron model algorithm:

1. Initialize randomly weights.
2. For each training sample, compute the weighted sum of the inputs.

$$\text{weighted sum} = \sum_{j=1}^{n} w_j x_j + b$$

3. Pass the sum through the activation function to make a prediction.

$$\hat{y} = \text{sign}(\text{weighted sum})$$

4. Update the weights based on the prediction error (difference between predicted and actual

$$w_j = w_j + \eta(y_i - \hat{y})x_j$$

   labels).
5. Repeat the process for several epochs (until convergence, however we modelled for a fixed set of epochs)

Hyperparameter Tuning with Grid Search

For training the Perceptron, certain hyperparameters that influence the model's performance were tuned. I focused on the following hyperparameters:

- **Learning Rate**: Determines the size of the steps the model takes while adjusting the weights. The tested values are 0.01, 0.1, and 1 to find the optimal step size for convergence.
- **Epochs**: Defines the number of times the model will iterate over the entire training data. Considered different values: 50, 100, 200, and 500.

- **Activation Function**: The **sign function** was used.

To find the best combination of the hyperparameters (learning rate and epochs), I used 5-fold cross-validation. This method splits the training data into five parts, training the model on four parts and validating it on the remaining one, repeating this process for each fold. This technique ensures that the model's performance is assessed robustly.

After selecting the best hyperparameters, I trained the Perceptron model using the full training set and evaluated it on the test set. The model's performance was assessed based on:

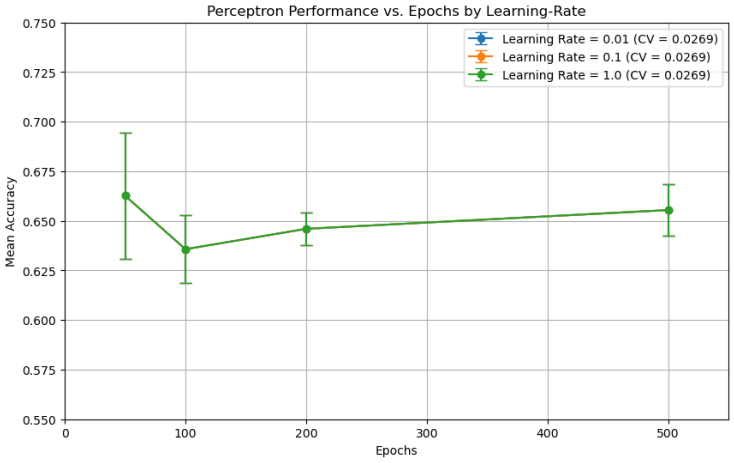
Accuracy: The percentage of correctly classified instances in the test set.

Loss: A measure of how well the predicted values match the actual values.

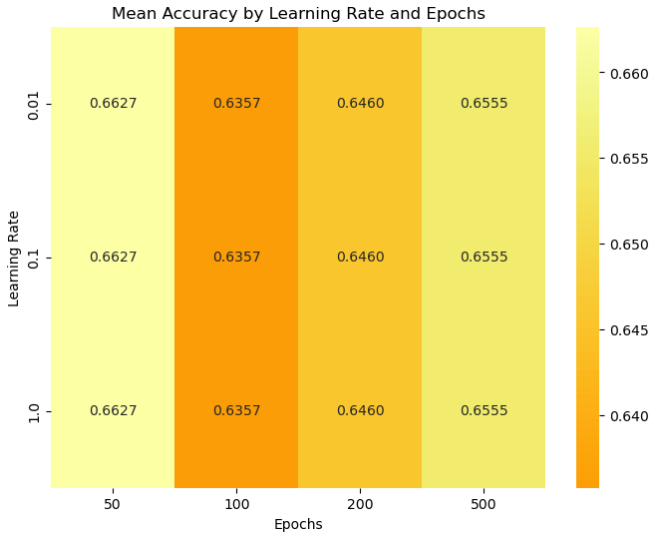
Results and Visualizations

To understand the impact of **learning rate** and **epochs** on the model's performance, I generated several visualizations.

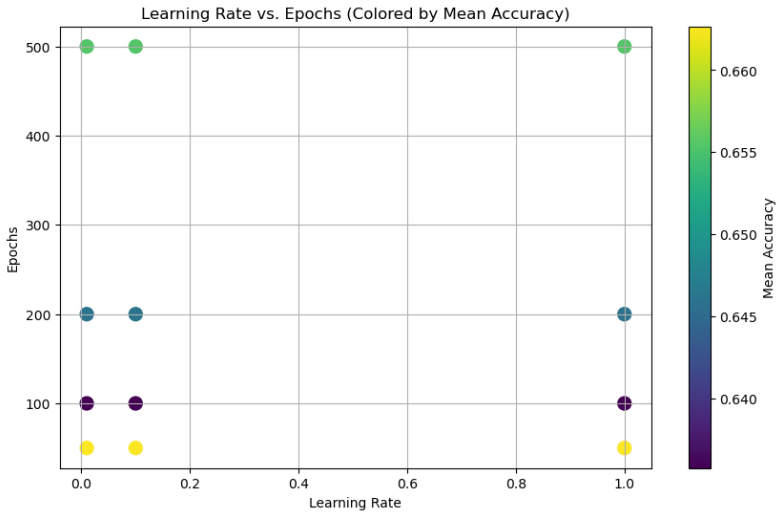
- **Mean Accuracy vs. Epochs for Different Learning Rates**: This plot shows how the accuracy of the Perceptron model changes as the number of epochs increases, with different learning rates. It also includes the standard deviation to visualize the variability in performance across cross-validation folds. Further, for the three different learning rates the accuracy was close enough to noe be separately seen on the plot below.



- **Heatmap of Mean Accuracy**: A heatmap was created to show the accuracy for different combinations of learning rates and epochs, providing a clear view of the optimal configurations.



- **Learning Rate vs. Epochs (Colored by Mean Accuracy)**: This scatter plot visually represents the relationship between learning rates, epochs, and the resulting mean accuracy, allowing us to identify how these hyperparameters interact to affect model performance.



Model Results

Best Hyperparameters (Training Set):

Learning Rate: 0.01; Epochs: 50; Activation Function: sign; Training Accuracy: **66.27%**

Test Accuracy for Perceptron with Best Hyperparameters: **50.00%**

Loss for Perceptron with Best Hyperparameters: 0.5000

This indicates that the model performed no better than random guessing on the test set, suggesting that the chosen hyperparameters did not lead to effective classification.

SVM Model with Pegasos objective Function

Support Vector Machines (SVM) with the Pegasos algorithm for classification tasks is a machine learning method that learns the optimal hyperplane to separate classes. The Pegasos algorithm is an efficient way to solve the optimization problem for SVMs using stochastic gradient descent, especially for large-scale datasets.

The Pegasos SVM was trained and evaluated using a grid search approach over two hyperparameters:

1. **Lambda** (λ): Regularization parameter that controls the complexity of the model.
2. **Epochs**: Number of iterations to update the model's weights during training.

Algorithm for Pegasos

Parameters: number T of rounds, regularization coefficient $\lambda > 0$
Input: Training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in \mathbb{R}^d \times \{-1, 1\}$
Set $\boldsymbol{w}_1 = \mathbf{0}$
For $t = 1, \ldots, T$

1. Draw uniformly at random an element $(\boldsymbol{x}_{Z_t}, y_{Z_t})$ from the training set
2. Set $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla \ell_{Z_t}(\boldsymbol{w}_t)$

Output: $\overline{\boldsymbol{w}} = \frac{1}{T}(\boldsymbol{w}_1 + \cdots + \boldsymbol{w}_T)$.

Hyperparameter Grid Search:

A grid search was performed over the following values: Lambda: [0.01, 0.1, 1]; Epochs: [50, 100, 200, 500]

The model was validated using **5-fold cross-validation** to estimate performance reliably across different hyperparameter settings. The results showed the following:

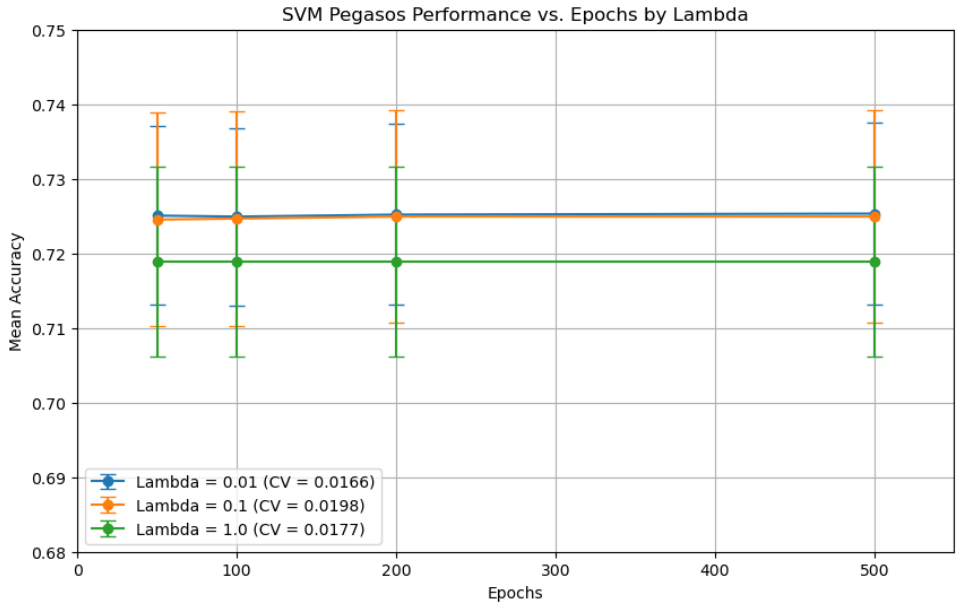
Best Hyperparameters: lambda = 0.01, **Epochs = 500**, Best Cross-Validation Score: **72.53%.**

Test Accuracy with Best Hyperparameters: **74.76%** (higher as compared to the perceptron).

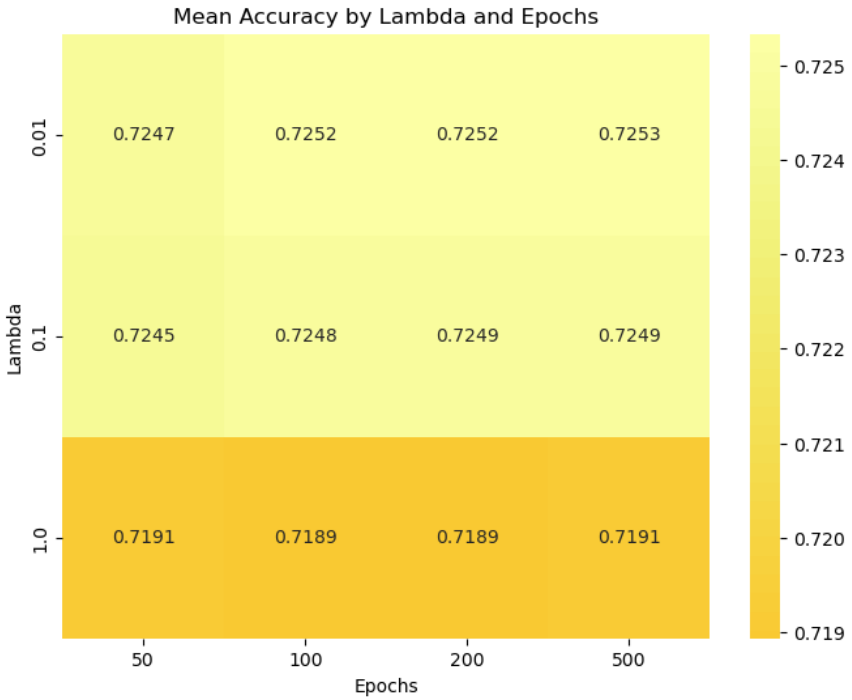
Loss: 0.2524 (indicating the model's accuracy is high, with a relatively low error).

Model Performance and Visualization:

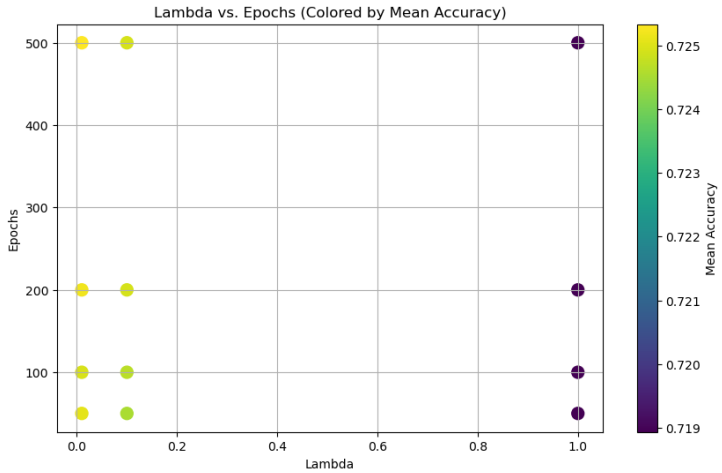
1. **Mean Accuracy vs. Epochs for Different Lambda Values**: A plot was generated to examine how the **mean accuracy** varied with the number of epochs for different values of λ. The results suggested that the optimal performance was achieved at λ=0.01, with increasing epochs showing a slight increase in accuracy.



2. **Heatmap of Mean Accuracy by Lambda and Epochs**: A heatmap was created to visualize the relationship between lambda and epochs on the mean accuracy. The best performance appeared when λ=0.01 and 500 epochs were used, as reflected in the color intensity.



3. **Scatter Plot of Lambda vs. Epochs Colored by Mean Accuracy**: A scatter plot was used to illustrate how lambda and epochs interact with mean accuracy. The plot indicated that a higher value of epochs with lower lambda resulted in better performance.



Model Results:

The SVM with Pegasos demonstrated a better performance with the best hyperparameters yielding a test accuracy of **74.76%.**, confirming that the model achieved good separation between the classes, though there is room for improvement. Furthermore, the test accuracy is in the range of training accuracy of **72.53%** which also signifies that the model is not overfitting.

Regularized Logistic Classification

Logistic Regression with regularization to add a penalty term reduces the complexity of the model. The Pegasos algorithm is used for solving the regularized logistic regression problem using stochastic gradient descent (SGD).

The regularization used is the L2 regularization term and with the Pegasos algorithm for optimization. The model is tuned using a grid search for different hyperparameters, including regularization strength, lambda (λ) and number of epochs.

Objective Function: $J(w) = \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2$

where:

- w are the model weights,
- xi are the input features,
- yi are the target labels,
- λ is the regularization strength.

Optimization with Pegasos:

- The Pegasos algorithm efficiently optimizes the above objective using stochastic gradient descent (SGD).
- During each iteration, we update the weights as:

  $$w_{t+1} = w_t - \eta_t \nabla_w J(w_t)$$

  where η is the learning rate, and is calculated as $\eta_t = \frac{1}{\lambda t}$

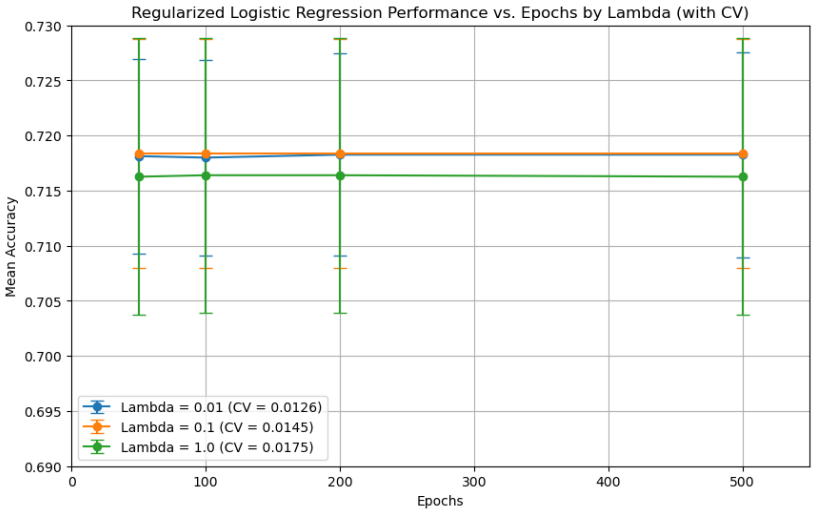
Hyperparameter Tuning:

A grid search was used to select the best combination of hyperparameters, specifically lambda (regularization strength) and epochs (the number of training iterations). The values for lambda tested are: [0.01, 0.1, 1] and the values for epochs are: [50, 100, 200, 500].

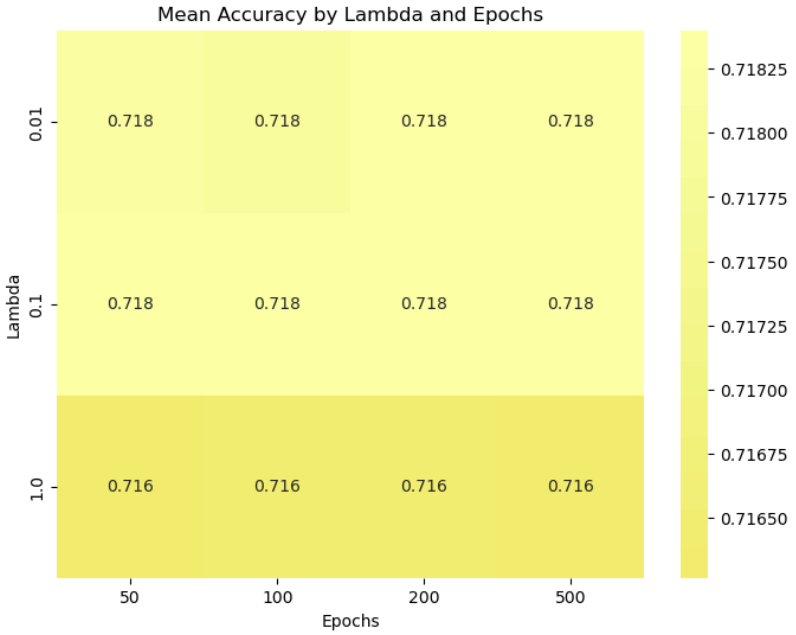
A 5-fold cross-validation was performed to evaluate model performance for different hyperparameter combinations to ensures that the model generalizes well across various subsets of the data.

Model Performance and Visualization:

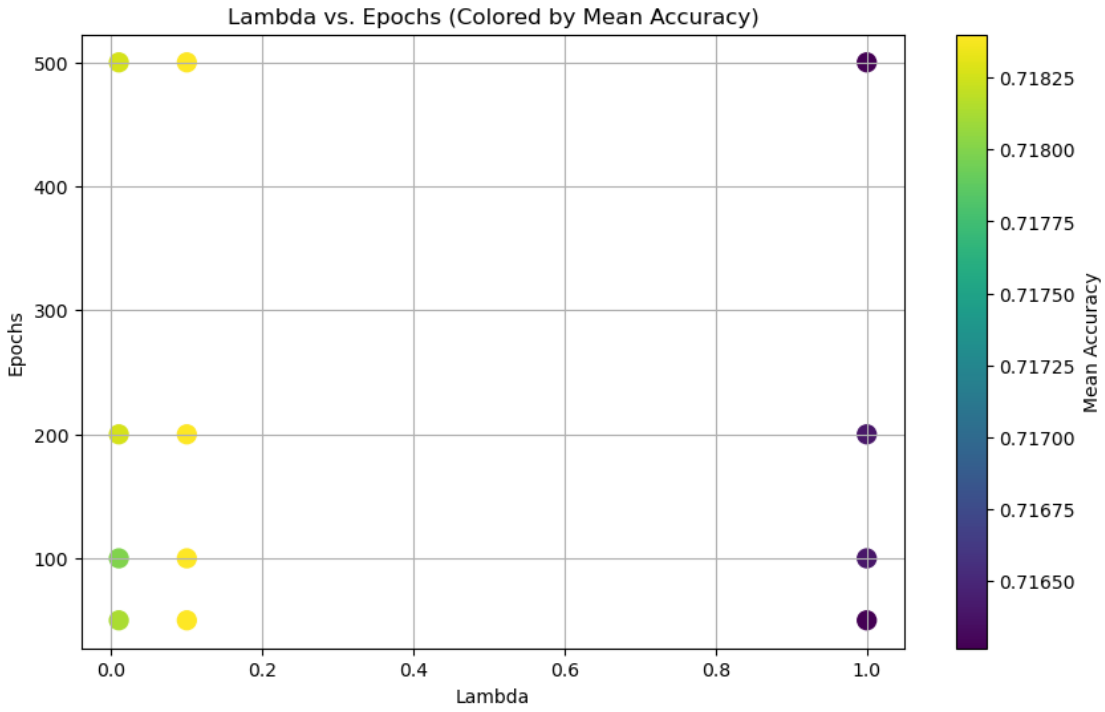
1. **Mean Accuracy vs. Epochs for Different Lambda Values:** This plot shows how the accuracy changes with respect to the number of epochs for different regularization strengths (lambda). Lambda = 0.1 achieves the highest accuracy across all epoch values.



2. **Heatmap of Mean Accuracy by Lambda and Epochs:** The heatmap was created to show how the accuracy varies with both lambda and epochs. This allows for a clearer visual representation of which combination of parameters yields the best results.



3. Scatter Plot of Lambda vs. Epochs: The scatter plot shows the relationship between lambda, epochs, and mean accuracy. The color gradient on the scatter plot represents the accuracy, with higher values colored differently for easy comparison.



Model Results:

After selecting the best hyperparameters, we train the Logistic Regression Pegasos model with:

Lambda = 0.1; **Epochs** = 50, with Training accuracy of **71.84%.**

Test Accuracy: The model achieves an accuracy of **74.28%** on the test set, which is a strong result for the problem at hand. Further, it is almost equal to the one achieved by SVM i.e. **74.76%**.

Polynomial Expansion and Model Weights Comparison

To further strengthen the performance of models, the features were expanded to degree 2 polynomial, on the training and testing datasets. The transformation effectively generates new features that include squares and interaction terms of the original features. This allows the models to learn more complex decision boundaries, particularly for non-linear classification tasks. The expanded feature sets were then used to train the models and evaluate their performance.

Model Training and Cross-Validation with Polynomial Features

With the polynomial features in place, we proceeded with cross-validation to find the best hyperparameters for each model. The following three models were again trained and evaluated:

- Perceptron
- Pegasos SVM
- Logistic Regression Pegasos

After performing a grid search for hyperparameter tuning, which allowed identification of the optimal combination of parameters for each model. The best-performing hyperparameters for each model with polynomial features were as follows:

Perceptron: Learning rate = 0.01, Epochs = 50, Activation function = 'sign', Training Accuracy: 92.35%, and *Test Accuracy: 71.72% (shows overfitting)*.

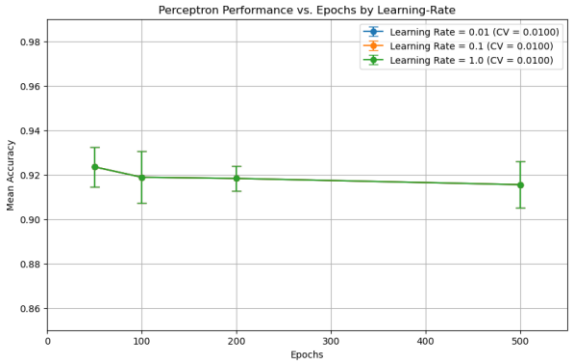
Pegasos SVM: λ (lambda) = 0.01, Epochs = 100, Training Accuracy: 94.32%, and *Test Accuracy: 94.16% (no overfitting).*

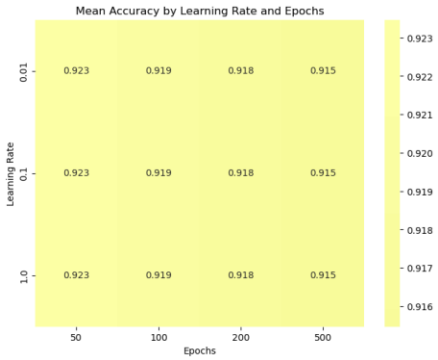
Logistic Regression Pegasos: λ (lambda) = 0.01, Epochs = 500, Training Accuracy: 93.88%, and *Test Accuracy: 93.72% (no overfitting).*

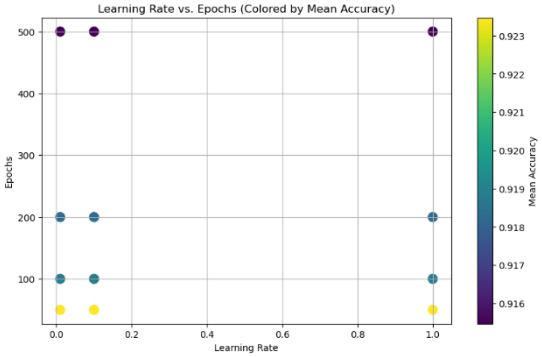
The models were then trained with the identified hyperparameters on the polynomial expanded features.

Visualization Plots:

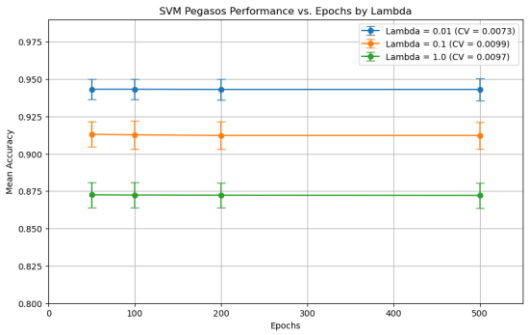
1) Perceptron

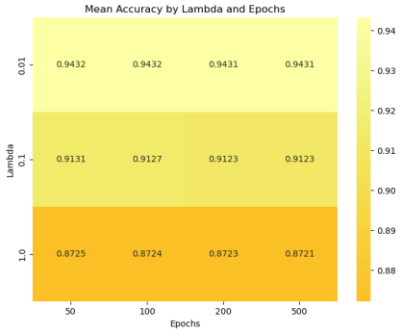


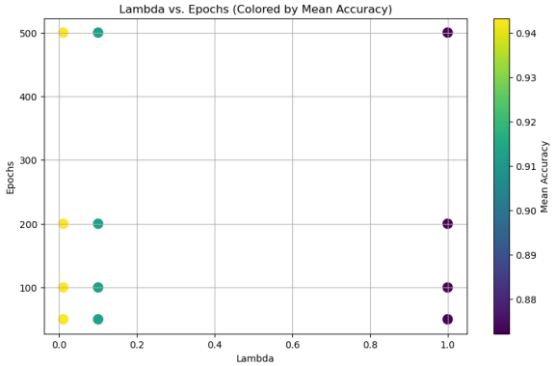




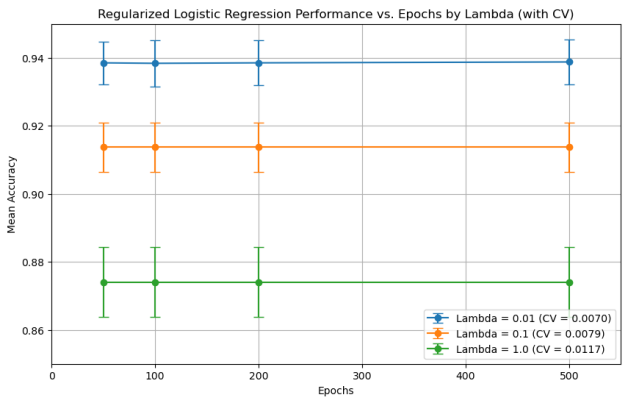
2) SVM Pegasos

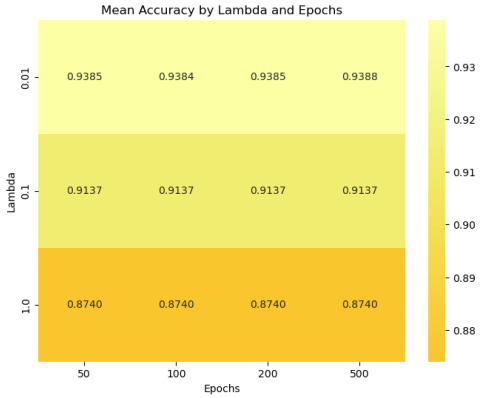


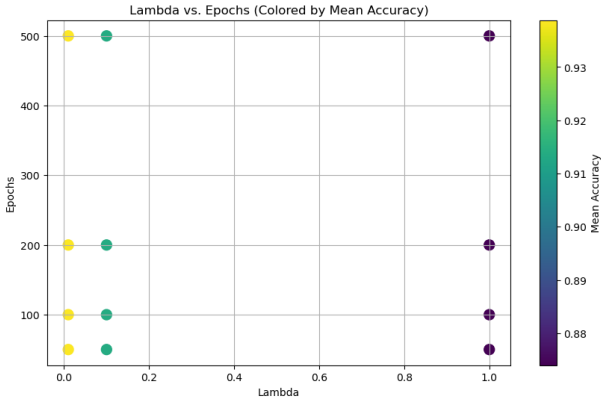




3) Logistic Regression Pegasos







Performance Evaluation

The models on the test set using accuracy metrics were evaluated. The results demonstrated that the Pegasos SVM with polynomial features achieved the highest test accuracy of 94.16%. In contrast, the Perceptron performed less favorably with a test accuracy of 71.72%, while the Logistic Regression Pegasos model performed similarly to the Pegasos SVM with a test accuracy of 93.72%.

Weight Comparison Across Models

After training the models, the learned weights of the models were analyzed to understand how each model is utilizing the features, particularly after the expansion to polynomial features. The comparison of weights offers insight into how each model has adjusted its parameters based on the complexity of the feature space.

Below is the summary of the weights learned by each model:

- **Perceptron (without polynomial features)**: The weights are relatively small in magnitude that reflects a linear decision boundary.
    - Weights for Perceptron Model: [0.01569562, -0.027697, 0.01083035, -0.00755328, -0.00398106, -0.00305679, 0.00216763, 0.05682532]
- **Pegasos SVM (without polynomial features)**: Larger weights compared to the Perceptron, indicating that the model has learned a more complex decision boundary.
    - Weights for Pegasos SVM Model: [0.12745166, -0.01306419, 0.10088369, -0.16098056, 0.16967422, 0.15034592, 0.36987858, 0.07458134, -0.04147306]
- **Logistic Regression Pegasos (without polynomial features)**: The weights are similar in size to those of the Pegasos SVM model, signifying a linear decision boundary learned through logistic regression.
    - Weights for Logistic Regression Pegasos (without polynomial features) : [0.20402015, -0.01651521, 0.13513333, -0.25183331, 0.20339681, 0.24643115, 0.49605225, 0.10277368].
- **Perceptron with Polynomial Features**: The inclusion of polynomial features led to a significant increase in the number of features and larger weights. The model is now capturing higher-order feature interactions, leading to a more complex decision boundary.
    - Weights for Perceptron with Polynomial Features Model: [0.38561539, -0.2141344, 0.11160008, -0.22291356, 0.6100209, 0.15170631, 1.40105223, 0.56909521, 0.09454923, 0.41643074, 0.16854106, 0.08116751, 0.22159622, 0.20717086, 0.22771606, 0.22108504, 0.15106096, -0.13848109, 0.00362564, -0.21468914, 0.08949031, 0.9047804, 0.16008141, -0.01616914, 0.12997756, 0.08077478, 0.1109016, 0.30285804, 2.45490211, 0.1057403, -0.023904, 0.10141754, -0.01990827, 0.31580623, 0.23048054, -0.15781175, -0.83038335, 0.17326316, -0.13934099, -0.05324466, -0.09774976, 0.20532879, 0.0240804, -0.02849806]
- **Pegasos SVM with Polynomial Features**: This model exhibits much larger weights than its non-polynomial counterpart, as it is now learning decision boundaries in a higher-dimensional space created by the polynomial features.
    - Weights for Pegasos SVM with Polynomial Features Model: [1.98904396e-01, 5.07909570e-02, 1.11385402e-01, -2.06898739e-01, 2.30975816e-01, 2.02781856e-01, 5.80770379e-01, 2.26143427e-01, -3.91766986e-02, 8.40442836e-02, -1.82879574e-02, -5.71585319e-02, 1.64293246e-03, -2.62276470e-03, -1.41687734e-02, 6.97130459e-02, 5.46407121e-02, -

2.15065965e-02, -7.51762390e-02, 2.94489658e-03, 1.74872855e-02, 3.06172647e-01, -1.72359440e-02, -3.26576193e-02, 3.17546998e-02, -1.76000334e-02, 3.57501676e-03, -3.26330367e-03, 9.51755932e-01, 1.05878483e-02, 2.54940271e-02, 1.25769522e-02, -6.13642203e-03, 1.92999945e-02, 3.45533532e-02, 5.42403980e-03, -3.03906856e-01, -8.37445365e-04, 4.75780589e-02, 2.78166676e-02, -1.30894358e-02, 4.41654993e-03, 7.02716748e-03, 2.15685221e-02, -1.81023026e-01]

- **Logistic Regression Pegasos with Polynomial Features**: Similar to the Pegasos SVM, the Logistic Regression Pegasos model shows an increase in the magnitude of its weights, indicating that polynomial features are allowing the model to learn more intricate relationships.
    - Weights for Logistic Regression Pegasos with Polynomial Features Model: [0.56567911, 0.12573669, 0.31274465, -0.56018912, 0.60737417, 0.55797367 1.58171597, 0.59324886, -0.15603439, 0.1736478, -0.14709076, -0.20511464, -0.12364992, -0.06278594, -0.11926034, 0.1496043, 0.15417807, -0.06798971, -0.15589792, -0.04391742, 0.00863417, 0.83018217, -0.00425645, -0.09916723, 0.09355088, -0.0356375, 0.04565385, 0.02533906, 2.54480874, 0.05625204, 0.05763418, 0.02258211, -0.02788301, 0.05759986, 0.12101649, 0.05929677, -0.8451857, 0.01395313, 0.09248067, 0.04170843, -0.01546614, 0.01988202, 0.05040674, 0.09259382]

The SVM with Pegasos model, after hyperparameter optimization via grid search, performed well with a test accuracy of 94.16%. The model benefited from tuning the lambda and epochs, with the best performance occurring at a lambda of 0.01 and 100 epochs. Further refinement of hyperparameters or testing on additional datasets could improve the model's performance.

Kernelized Perceptron (Polynomial and Gaussian Kernels)

The Kernelized Perceptron is an extension of the standard perceptron algorithm, utilizing kernel functions to transform the input space into a higher-dimensional feature space.

The two kernels evaluated in this report are:

Polynomial Kernel: This kernel computes similarity based on a polynomial function of the dot product.

RBF (Gaussian) Kernel: This kernel measures similarity by mapping data to a higher-dimensional space using a Gaussian function.

Parameter Tuning and Cross-Validation:

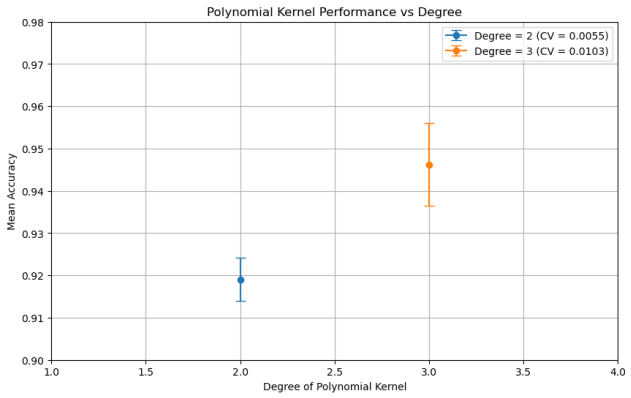
Polynomial Kernel:

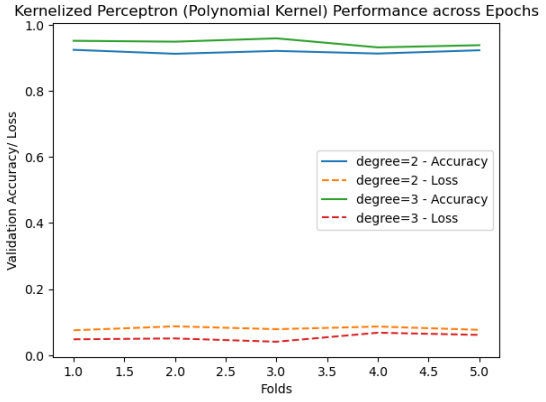
Performing a custom cross-validation with a grid search to optimize the degree of the polynomial kernel. The parameter grid used for the polynomial kernel was: “degree = [2, 3]”;

Cross-validation Results:

Degree 2: Mean Accuracy: 91.91%, Standard Deviation: 0.51%, Coefficient of Variation (CV): 0.55%

Degree 3: Mean Accuracy: 94.63%, Standard Deviation: 0.97%, Coefficient of Variation (CV): 1.03% *(Best Parameters)*





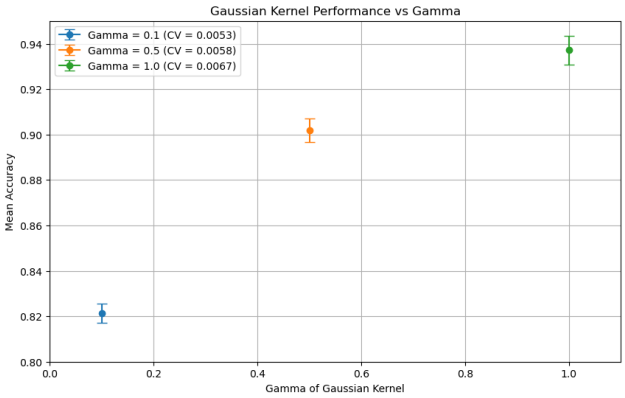
Evaluation on Test Set: Test Accuracy is 78.96%, which shows that the model was overfitting on the training set.

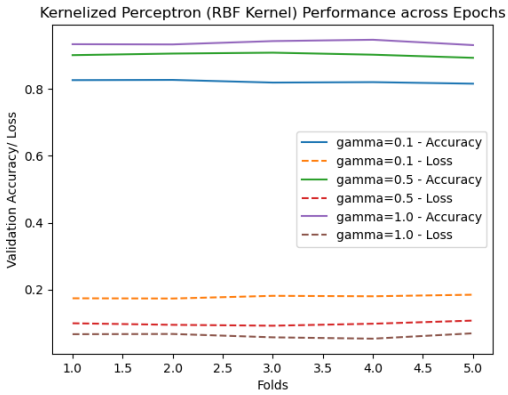
RBF Kernel:

Performing a custom cross-validation to optimize the gamma parameter for the RBF kernel. The parameter grid used for the RBF kernel was, gamma = [0.1, 0.5, 1.0]

Cross-validation Results:

- **Gamma 0.1**: Mean Accuracy: 82.13%, Standard Deviation: 0.44%, Coefficient of Variation (CV): 0.53%
- **Gamma 0.5**: Mean Accuracy: 90.17%, Standard Deviation: 0.52%, Coefficient of Variation (CV): 0.58%
- **Gamma 1.0**: Mean Accuracy: 93.72%, Standard Deviation: 0.63%, Coefficient of Variation (CV): 0.67% *(Best Parameters)*.





Evaluation on Test Set: Test Accuracy is 89.72%, which shows that the model was slightly overfitting on the training set.

Both Polynomial and RBF kernels showed strong performance with high accuracy during cross-validation. However, there was a noticeable drop in performance when evaluated on the test set, for the Polynomial kernel, where the test accuracy was lower than the cross-validation score. This indicates a potential overfitting issue, particularly with the Polynomial kernel at degree=3.

Kernelized SVM with Pegasos (Polynomial and Gaussian Kernels)

In Kernelized SVMs, a kernel function maps the data into a higher-dimensional space, enabling the SVM to find a non-linear separating hyperplane. The Kernelized Pegasos algorithm is a variant of the Pegasos SVM, which uses a stochastic gradient descent-based approach for optimization.

For the kernelized SVM with Pegasos objective function, the given research paper was used with the following error being corrected in the algorithm, $y_{i_t}\left(\frac{1}{\lambda t}\sum_{j=1}^{|S|}\alpha_t[j]y_j K(x_{i_t},x_j)\right)<1$,

Kernelized SVM Pegasos with Polynomial Kernels:

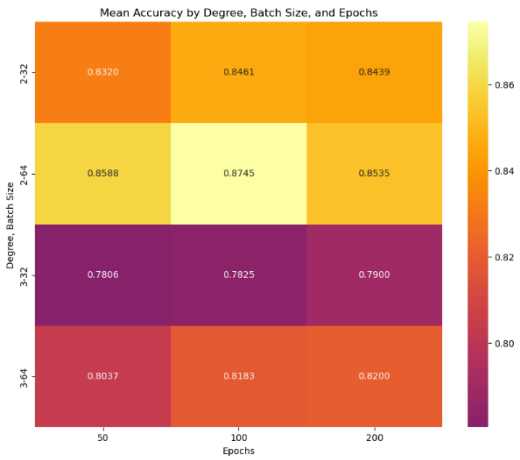
<u>Hyperparameter Tuning and Cross Validation:</u>

Regularization parameter (λ): Controls the trade-off between the margin size and classification error, [0.01, 0.1, 1].

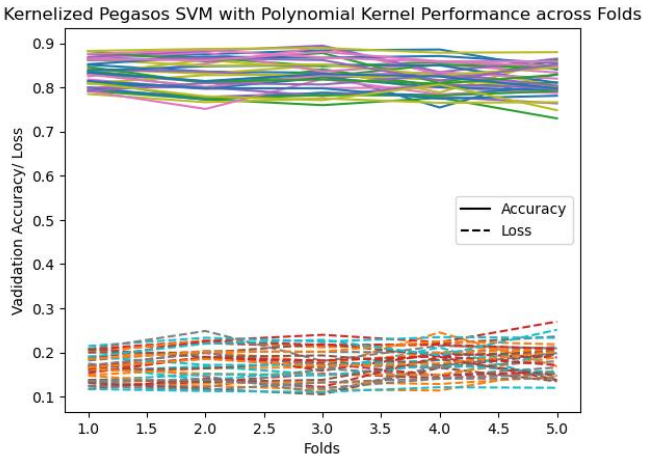
Number of iterations (T): The number of times the entire dataset is passed through the algorithm, [50, 100, 200].

Batch size: The number of data points used in each update during stochastic gradient descent, [32, 64].

Degree: The degree of polynomial used. [2, 3].



Heatmap



Accuracy & Loss across different folds (training)

Best Parameters in Training Set:

λ = 1, 'T' = 100, 'degree' = 2, 'batch_size' = 64, with Training Accuracy = **88.36%.**

Evaluation on Test Set:

Test Accuracy for Kernelized Pegasos SVM Polynomial Kernel: **86.84%** which is in line with the training accuracy this showing no overfitting.

Kernelized SVM Pegasos with Gaussian Kernels:

Hyperparameter Tuning and Cross Validation:

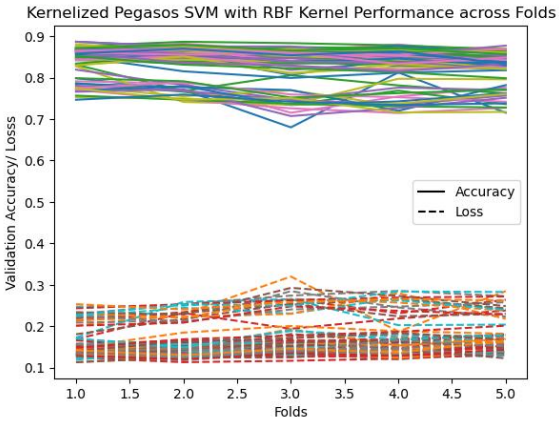
Regularization parameter (λ): Controls the trade-off between the margin size and classification error, [0.01, 0.1, 1].

Number of iterations (T): The number of times the entire dataset is passed through the algorithm, [50, 100, 200].

Batch size: The number of data points used in each update during stochastic gradient descent, [32, 64].

Gamma: The gamma parameter for the gaussian kernel, [0.1, 0.5, 1, 1.5].

Heatmap



Accuracy & Loss across different folds (training)

Best Parameters in Training Set:

λ = 0.01, 'T' = 200, ‘gamma’ = 1, 'batch_size' = 64, with Training Accuracy = **87.84%**.

Evaluation on Test Set:

Test Accuracy for Kernelized Pegasos SVM Gaussian Kernel: **89.12%** which is in line with the training accuracy (even better) this showing no overfitting.

Conclusion

The below table shows the comparison of the different models implemented in the project:

S.No.	Model	Training Accuracy	Test Accuracy	Computation Time (Training) (minutes) (approx)	Comment (s)
1	SVM + Polynomial Expansion	94.32%	94.16%	10.00	0.47% Higher Accuracy than Reg. Log + Poly Expansion
2	Regularized Logistic + Polynomial Expansion	93.88%	93.72%	1.30	87% lower Computation Time than SVM + Poly Exp.
3	Kernelized Perceptron (RBF)	93.72%	89.72%	2.50	-
4	Kernelized SVM (RBF)	87.84%	89.12%	546.00	-
5	Kernelized SVM (Poly)	88.36%	86.84%	43.00	-
6	Kernelized Perceptron (Poly)	94.63%	78.96%	1.50	-
7	SVM	72.53%	74.76%	11.00	-
8	Regularized Logistic	71.84%	74.28%	3.00	-
9	Perceptron + Polynomial Expansion	92.35%	71.72%	7.50	-
10	Perceptron	66.27%	50%	4.20	-

The models are arranged as per the highest accuracy achieved over the test dataset. It is found that the SVM with polynomial expansion of degree=2 worked the best, however on closer inspection, the regularized logistic regression with polynomial expansion of degree=2 also achieves the similar accuracy, but with a contrastingly less computation time (87% less than that of SVM). Thus, the regularized logistic regression model with degree=2 is chosen as the best working model.

Further work

To further strengthen the results of the models, an extensive training for different hyperparameters and cross-validation folds can be performed. Further, other classification techniques such as decision trees can also be used to cross-validate the results from these models.

References

1. https://www.statlearning.com/
2. https://www.google.com/url?q=https://home.ttic.edu/~nati/Publications/PegasosMPB.pdf&sa=D&source=editors&ust=1722278116861191&usg=AOvVaw1YHDjhMouYrQO9PDCTyVST
3. https://www.youtube.com/playlist?list=PLKnIA16_Rmvbr7zKYQuBfsVkjoLcJgxHH
4. https://scikit-learn.org/stable/supervised_learning.html

Appendix

The markdown files for the project are uploaded on the public repository, Github.

Link: https://github.com/sky-akash/ML-Kernel-Based-Classification