Assignment 2: UMC 203 AI and ML

Ashwin K.M. (23882)

28/03/2024
Academic Year 2025-2026

Contents

1 SVMs and Perceptron Algorithm

Tasks

1.1 Perceptron Algorithm

Problem Statement

Run the perceptron algorithm on your data. Report whether it converges, or appears not to. If it doesn't seem to converge, make certain that you are reasonably sure.

Implementation Details

The perceptron algorithm was implemented to determine whether it converges on the given dataset. The weight vector w^* was initialized to a zero vector of appropriate dimensions, and the algorithm iteratively updated w^* based on misclassified instances.

At each iteration, the feature matrix and label vector were extracted from the training data. The decision function was computed as the dot product between the feature matrix and w^*, multiplied by the corresponding labels. Instances where the decision function was non-positive were identified as misclassified. The weight vector was then updated using these misclassified instances.

To assess convergence, the misclassification rate was computed at each iteration as the ratio of misclassified instances to the total number of training samples. A deque structure was employed to store the misclassification rates of the last 20 iterations. Convergence was assessed using two criteria:

- If the misclassification rate reached zero, the perceptron was deemed to have converged.
- If the range of misclassification rates within the last 20 iterations remained below a predefined tolerance, this indicated a potential cyclic behavior in weight updates, signifying that the perceptron would not converge.

A rolling average smoothing function was applied to the misclassification rate history to visualize trends. The perceptron was trained for a maximum of one million iterations, and training was terminated early upon convergence or loop detection. The final outcome included the trained weight vector, the convergence status, and the total number of iterations taken.

1.2 Kernelized Support Vector Machine

Problem Statement

Repeat the previous construction, but with the Gaussian kernel this time. Choose your hyperparameters such that non-separability is no longer an issue, and your decision boundary is consistent with the training data's labels.

Implementation Details

To construct a Kernelized Support Vector Machine (SVM) using the Gaussian kernel, we implement a dual formulation of the SVM. Our objective is to tune the hyperparameters such that non-separability is no longer an issue and the decision boundary aligns with the training data labels.

We employ a grid search methodology to systematically explore reasonable values of the regularization parameter C and the kernel parameter γ, assessing their impact on misclassification rates. The procedure is as follows:

1. We define the Gaussian (Radial Basis Function) kernel as:

$$K(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|^2)$$

   where γ controls the influence of individual training samples.

2. The SVM's dual problem is solved for each combination of C and γ from predefined sets:

$$C \in \{0.1, 1.0, 10.0, 100.0, 1000.0\}, \quad \gamma \in \{0.1, 1.0, 5, 10.0, 20, 30, 50.0, 100.0\}.$$

3. The Lagrange multipliers α are extracted from the dual solution. Support vectors are identified as those samples where α satisfies:

$$1 \times 10^{-5} < \alpha_i < C - 1 \times 10^{-5}.$$

4. The bias term b is computed by averaging over all support vectors:

$$b = \frac{1}{|S|} \sum_{i \in S} \left(y_i - \sum_{j \in S} \alpha_j y_j K(x_j, x_i) \right),$$

   where S is the set of support vectors.

5. The trained SVM model is used to classify the training set, and misclassified samples are identified as those for which:

$$y_i \left(\sum_{j \in S} \alpha_j y_j K(x_j, x_i) + b \right) \leq 0.$$

6. The misclassification rate is computed for each pair of C and γ, and the best-performing configuration is selected.

Following this process, we obtain the optimal hyperparameters:

$$C = 100, \quad \gamma = 30.$$

These values yield the lowest misclassification rate i.e 0, thereby ensuring the SVM constructs a robust decision boundary that effectively separates the training data.

Deliverables

1.3 Perceptron: Misclassification Rate vs Iterations

Problem Statement

A plot between misclassification rate and number of iterations for the perceptron algorithm as defined in Task 1. (2 marks)

Tolerance = 0.01

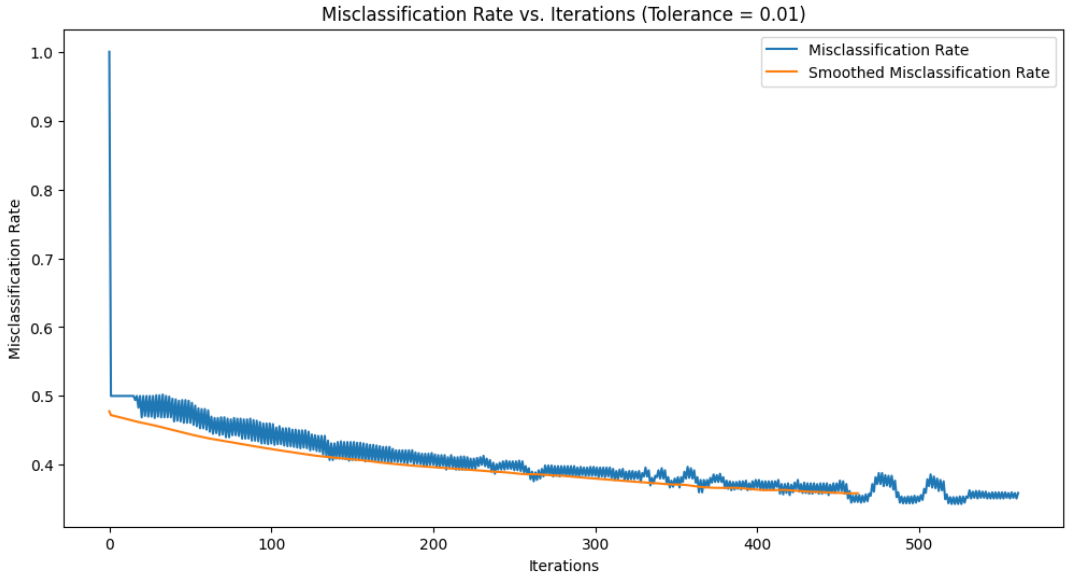

Figure 1: Misclassification Rate vs. Iterations for Tolerance = 0.01

Loop detected after 562 iterations. Stopping training. Perceptron did not converge.

Tolerance = 0.005

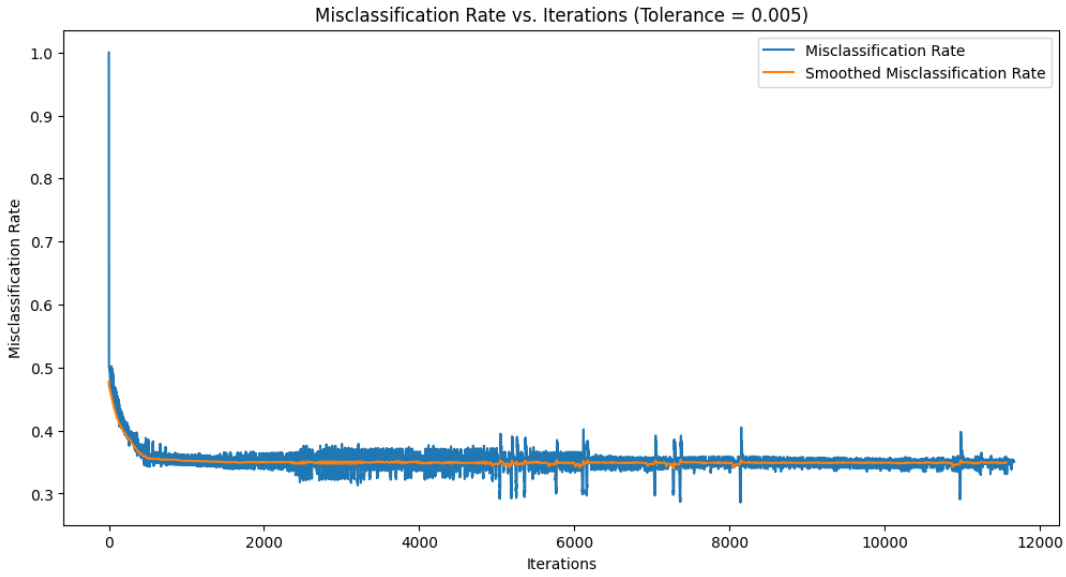

Figure 2: Misclassification Rate vs. Iterations for Tolerance = 0.005

Loop detected after 11,667 iterations. Stopping training. Perceptron did not converge.

Tolerance = 0.003

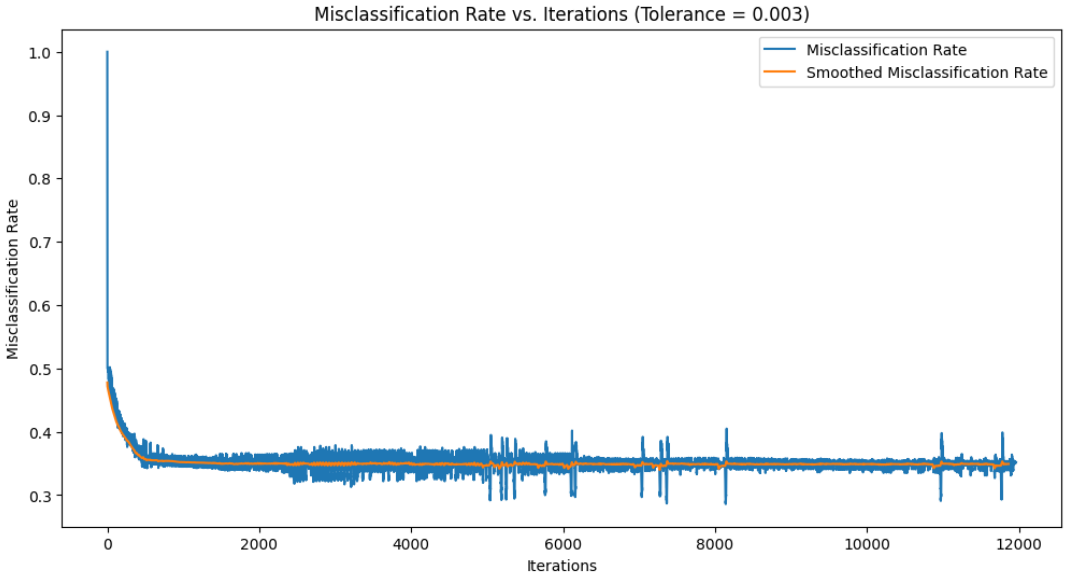

Figure 3: Misclassification Rate vs. Iterations for Tolerance = 0.003

Loop detected after 11,962 iterations. Stopping training. Perceptron did not converge.

Tolerance = 0.001

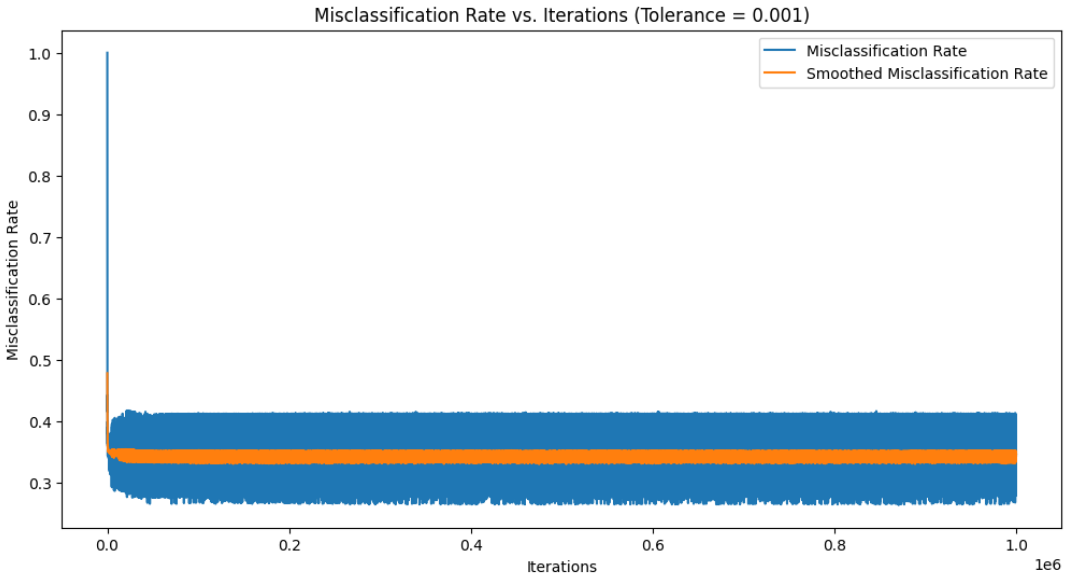

Figure 4: Misclassification Rate vs. Iterations for Tolerance = 0.001

No loop detected. Training continued until the maximum iteration limit was reached. Converged: False, Iterations: 1,000,000.

1.4 Primal vs Dual SVM: Computation Time

Problem Statement

Which, between the primal and dual, is solved faster for Task 2? Report the times taken for running both, and justify any patterns observed.

Theoretical Basis

The computational efficiency of solving an SVM problem depends significantly on the choice between the primal and dual formulations, particularly concerning the regularization parameter C.

Effect of C (Regularization Parameter):

- **Low C (e.g., 0.1, 1):**
  - More support vectors are retained, allowing for a wider margin and higher misclassification tolerance.
  - The optimization problem is more flexible, leading to lower computational time.
  - Higher bias, lower variance—better generalization at the cost of some misclassification.
- **High C (e.g., 10, 100):**
  - Fewer support vectors remain, enforcing a stricter margin with fewer misclassifications.
  - The optimization problem becomes harder to solve due to stricter constraints, increasing computational time.
  - Lower bias, higher variance—more prone to overfitting as it tries to classify every training point correctly.

Based on this theoretical understanding, we expect:

1. **Number of support vectors vs.** C: A decreasing trend as C increases.
2. **Computation time vs.** C: An increasing trend, as stricter constraints make optimization more computationally expensive.

Experimental Results

To validate the theoretical insights, we conducted an experiment to measure the average computation time for solving the SVM problem using both the primal and dual formulations. The optimization process was repeated for 20 iterations for each value of C, and the average time taken was recorded. The following graph illustrates the results.

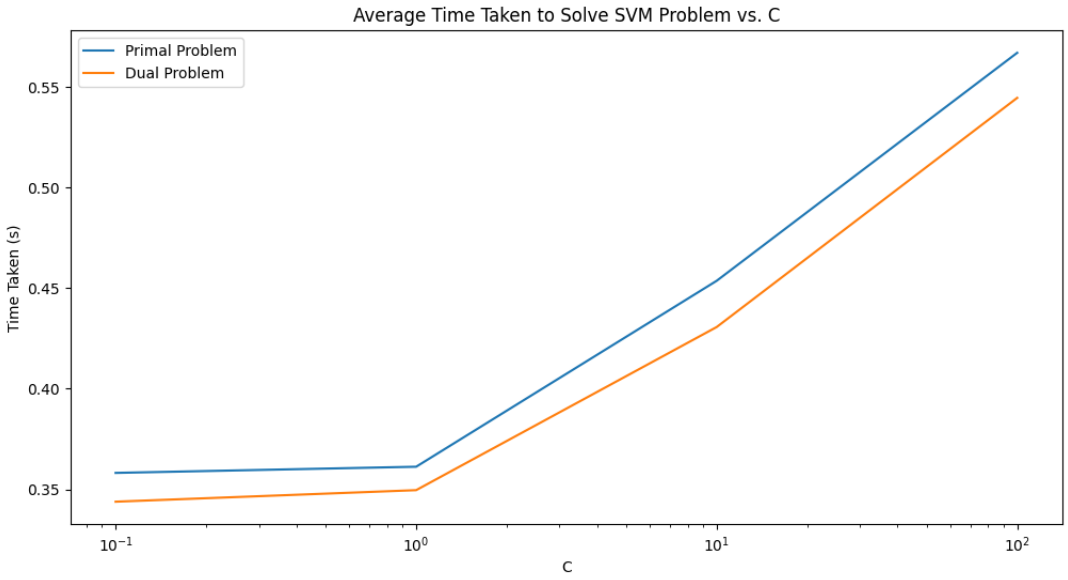

Figure 5: Average Time Taken to Solve SVM Problem vs. C for Primal and Dual Formulations.

Interpretation of Results

From Figure 5, we observe the following key trends:

- As C increases, the time taken to solve the SVM problem increases for both the primal and dual formulations. This aligns with our theoretical expectation that stricter constraints result in a more complex optimization problem.
- The dual formulation is consistently faster than the primal formulation across all values of C. This is because the number of variables in the optimization problem reduces from $d + N + 1$ in the primal formulation to just N in the dual formulation, where d is the feature dimension and N is the number of training samples. This reduction simplifies the optimization process, making the dual approach computationally more efficient.
- The gap in computation time between the primal and dual problems widens as C increases. This suggests that for large C, the dual formulation scales better than the primal approach.

Conclusion

The experimental findings confirm our theoretical predictions. As the regularization parameter C increases, the computational cost of solving the SVM problem rises due to stricter constraints. Moreover, the dual formulation remains computationally superior to the primal approach, particularly for large C. This highlights the practical advantage of solving SVMs in the dual space, especially when handling high-dimensional data.

1.5 Non-Separability Analysis

Problem Statement

The images that cause non-separability. (2 marks)

1.6 Kernelized SVM: Final Misclassification Rate

Problem Statement

The final misclassification rate for the kernelized SVM for Task 3. (2 marks)

Results

Following the hyperparameter tuning process, we obtain the optimal values:

$$C = 100, \quad \gamma = 30.$$

These values yield the lowest misclassification rate, which is 0, indicating that the kernelized SVM successfully constructs a robust decision boundary that perfectly separates the training data.

1.7 Perceptron Re-evaluation: Misclassification Rate vs Iterations

Problem Statement

A plot between misclassification rate and iterations for the perceptron for Task 4. (2 marks)

Analysis

The perceptron learning algorithm iteratively adjusts its decision boundary based on misclassified points. Initially, the misclassification rate is high due to random initialization and frequent updates, but it gradually decreases as the model refines its decision boundary.
The following plot illustrates the relationship between the misclassification rate and the number of iterations:

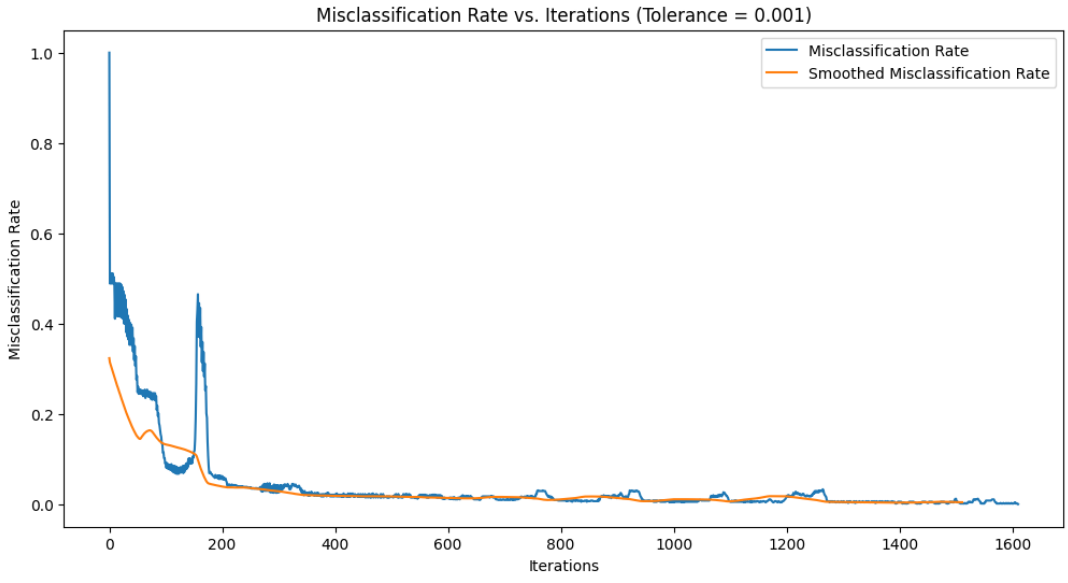

Figure 6: Misclassification Rate vs. Iterations for the Perceptron Algorithm (Tolerance = 0.001)

Observing the graph, we note:

- The misclassification rate starts close to 1, indicating frequent misclassifications in the early iterations.

- There are sharp drops in misclassification at certain points, reflecting significant updates to the decision boundary.
- As iterations progress, the misclassification rate stabilizes near zero, confirming convergence.
- The smoothed misclassification rate (orange curve) shows a general downward trend, despite fluctuations in the raw misclassification rate.

Effect of Removing Non-Separability

Previous SVM analysis identified sources of non-separability in the dataset. After addressing these, the perceptron successfully converges. The decreasing misclassification rate validates that the data has become linearly separable, allowing the perceptron to find a solution within a finite number of updates.
This result aligns with theoretical expectations: given linearly separable data, the perceptron algorithm is guaranteed to converge to a perfect classifier.

2 Logistic Regression, MLP, CNN PCA

Tasks

2.1 Training Configuration

All models in this study are trained using a standardized set of hyperparameters to ensure consistency and facilitate comparison. The chosen hyperparameters are as follows:

- **Loss Function:** Cross-Entropy Loss, which is standard for multi-class classification tasks.
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum.
- **Momentum:** 0.9, used to accelerate convergence and reduce oscillations.
- **Learning Rate:** 0.01.
- **Batch Size:** 32.
- **Number of Epochs:** 10.

Rationale for Using SGD with Momentum Over Adam

SGD with momentum is chosen over Adam because it provides better generalization in scenarios where datasets are large, and it helps avoid overfitting. While Adam adapts the learning rate per parameter, making it effective in cases of sparse gradients, SGD with momentum tends to be more robust for classification tasks where the gradient landscape is smoother. Furthermore, momentum helps reduce oscillations, allowing for more stable updates.

2.2 Multi-Layer Perceptron

Problem Statement

Construct and train an MLP that takes a flattened image as input (for example, if the image size is 28×28, then the input dimension of the MLP should be 784) and gives class probabilities as output. (1 mark)

Construction

The Multi-Layer Perceptron (MLP) model is constructed with a total of seven layers, consisting of the following:

- An input layer of dimension 784 (flattened 28×28 image).
- Five fully connected hidden layers, each containing 128 neurons.
- An output layer with 10 neurons, corresponding to the number of classes.
- The activation function used in all hidden layers is the Rectified Linear Unit (ReLU).

The fully connected layers introduce a total of:

- $784 \times 128 + 128 = 100480$ parameters in the first layer,
- $128 \times 128 + 128 = 16512$ parameters in each of the four subsequent hidden layers,
- $128 \times 10 + 10 = 1290$ parameters in the output layer.

Thus, the total number of trainable parameters in the network is:

$$100480 + (4 \times 16512) + 1290 = 183818$$

Evaluation

The trained model is tested on a validation sample, yielding the following results:

- **Predicted Class:** 0.
- **True Class:** 0.

2.3 Convolutional Neural Network

Problem Statement

Construct and train a CNN that takes a direct image as input and gives class probabilities as output. (1 mark)

Construction

The Convolutional Neural Network (CNN) model consists of the following layers:

- **Convolutional Layer 1:** 3×3 kernel, 8 output channels, input channel of 1 (grayscale image).
- **Max Pooling Layer 1:** 2×2 kernel, stride 2.
- **Convolutional Layer 2:** 3×3 kernel, 16 output channels.
- **Max Pooling Layer 2:** 2×2 kernel, stride 2.
- **Fully Connected Layer 1:** 16*5*5 neurons (flattened feature map) to 64 neurons.
- **Fully Connected Layer 2:** 64 neurons to 10 output neurons.
- **Activation Function:** Rectified Linear Unit (ReLU) is applied to all layers except the output.

- **Dropout:** 30% dropout is applied after the first fully connected layer to prevent overfitting.

The number of trainable parameters in the CNN model is calculated as follows:

- **Convolutional Layer 1:** $(3 \times 3 \times 1 + 1) \times 8 = 80$
- **Convolutional Layer 2:** $(3 \times 3 \times 8 + 1) \times 16 = 1168$
- **Fully Connected Layer 1:** $(16 \times 5 \times 5 + 1) \times 64 = 25664$
- **Fully Connected Layer 2:** $(64 + 1) \times 10 = 650$

Thus, the total number of trainable parameters in the network is:

$$80 + 1168 + 25664 + 650 = 27562$$

Evaluation

The trained CNN model is tested on a validation sample, yielding the following results:

- **Predicted Class:** 0.
- **True Class:** 0.

2.4 Principal Component Analysis

Problem Statement

Extract image features using PCA (which is a famous dimensionality reduction technique). (1 mark)

PCA Implementation

Principal Component Analysis (PCA) is performed to reduce the dimensionality of the input data while preserving essential variance. The steps involved in applying PCA are outlined below:

1. **Standardization:** The dataset is standardized by subtracting the mean and dividing by the standard deviation for each pixel across all training images:

$$X_{\text{standardized}} = \frac{X - \mu}{\sigma + 10^{-6}} \tag{1}$$

   where X is the original dataset, μ is the mean, and σ is the standard deviation.

2. **Covariance Matrix Computation:** The covariance matrix is computed as:

$$C = \frac{1}{n} X_{\text{standardized}}^T X_{\text{standardized}} \tag{2}$$

   where C captures the relationships between different pixel intensities.

3. **Eigen Decomposition:** The eigenvalues and corresponding eigenvectors of the covariance matrix are computed:

$$Cv_i = \lambda_i v_i \tag{3}$$

   where λ_i are the eigenvalues and v_i are the eigenvectors.

4. **Sorting Principal Components:** The eigenvectors are sorted in descending order based on their corresponding eigenvalues, ensuring that the most significant variance is retained.

5. **Dimensionality Reduction:** The top k eigenvectors (principal components) are selected to form the transformation matrix W_k, which is then used to project the original data:

$$X_{\text{PCA}} = X_{\text{standardized}} W_k \tag{4}$$

   where k is set to 128 in this implementation.

6. **Transformation of Validation Data:** The validation set undergoes the same standardization and projection using the transformation matrix W_k derived from the training set.

After applying PCA, the training data is transformed to shape $(N, 128)$, where N is the number of training samples, effectively reducing the feature space while retaining most of the variance.

2.5 MLP with PCA

Problem Statement

Repeat Task 1, but now use handcrafted features extracted using PCA as input and class probabilities as output. (1 mark)

Implementation Details

In this approach, a Multi-Layer Perceptron (MLP) is trained on handcrafted features extracted using PCA. The input dimensions remain the same as the number of selected principal components, and the training procedure follows the standard supervised learning pipeline.

MLP Construction and Training

A fully connected MLP is constructed with an input dimension of 128, matching the number of principal components retained.

Evaluation and Class Probability Estimation

Once training is complete, the model is evaluated on validation data. Given an input image, the model produces a probability distribution over the possible classes. The softmax function is applied to obtain the probability scores, and the class with the highest probability is selected as the predicted label.

For instance, when evaluating a single validation image, the model outputs the following probability distribution:

- **Predicted class:** 0
- **True class:** 0

2.6 Logistic Regression with PCA

Problem Statement

Train a Logistic Regression model for multi-class classification and also train 10 binary classifiers for each class by the one-vs-rest approach using PCA features. (1 mark)

Multinomial Logistic Regression with (PCA)

In this approach, a multinomial logistic regression model is trained on handcrafted features extracted using PCA. The input dimensions are set to match the number of selected principal components ($k = 128$), and the model is trained using a standard supervised learning procedure.

Model Construction and Training

Multinomial logistic regression is a generalized linear model that extends binary logistic regression to multi-class classification. It models the probability of an input belonging to one of C classes using a linear transformation followed by the softmax function:

$$P(y = c|x) = \frac{\exp(W_c x + b_c)}{\sum_{j=1}^{C} \exp(W_j x + b_j)} \tag{5}$$

where W_c and b_c are the weight vector and bias term for class c, and x is the input feature vector.

Evaluation and Class Probability Estimation

After training, the multinomial logistic regression model is evaluated on validation data. Given an input image, the model outputs a probability distribution over the possible classes using the softmax activation. The class with the highest probability is selected as the predicted label. For example, when evaluating a single validation image, the model produces the following probability distribution:

$$\begin{array}{rcccc} & 9.9686e-01 & 3.1475e-08 & 3.0595e-03 & 3.7043e-05 \\ \text{Model output probabilities:} & 6.1127e-07 & 1.6774e-05 & 1.6709e-07 & 1.1366e-06 \\ & 1.4937e-06 & 2.1103e-05 & & \end{array} \tag{6}$$

where the highest probability corresponds to class 0. The final classification results for the image are:

- **Predicted class:** 0
- **True class:** 0

Deliverables

2.7 Image Reconstruction using Principal Components

Problem Statement

Reconstruct an image of your choice using principal components (1,2,3,...) and conclude the results. (1 mark)

Reconstruction Methodology

Given a dataset of flattened image vectors X, PCA projects them onto a lower-dimensional subspace defined by the top k eigenvectors of the covariance matrix. This transformation can be mathematically expressed as:

$$X_{\text{PCA}} = X_{\text{standardized}} W_k \tag{7}$$

where W_k represents the matrix of the top k eigenvectors, and X_{PCA} contains the lower-dimensional representations of the original images.

To reconstruct an image from its PCA representation, we perform the inverse transformation:

$$X_{\text{reconstructed}} = X_{\text{PCA}} W_k^T \cdot \sigma + \mu \tag{8}$$

where:

- $X_{\text{reconstructed}}$ is the reconstructed image.
- W_k^T transforms the PCA features back to the original feature space.
- σ and μ are the standard deviation and mean used for standardization during preprocessing.

Comparison of Original and Reconstructed Image

The figure below presents a visual comparison between an original image and its reconstructed counterpart. The left panel displays the original grayscale image, while the right panel shows its reconstruction using the top $k = 128$ principal components.

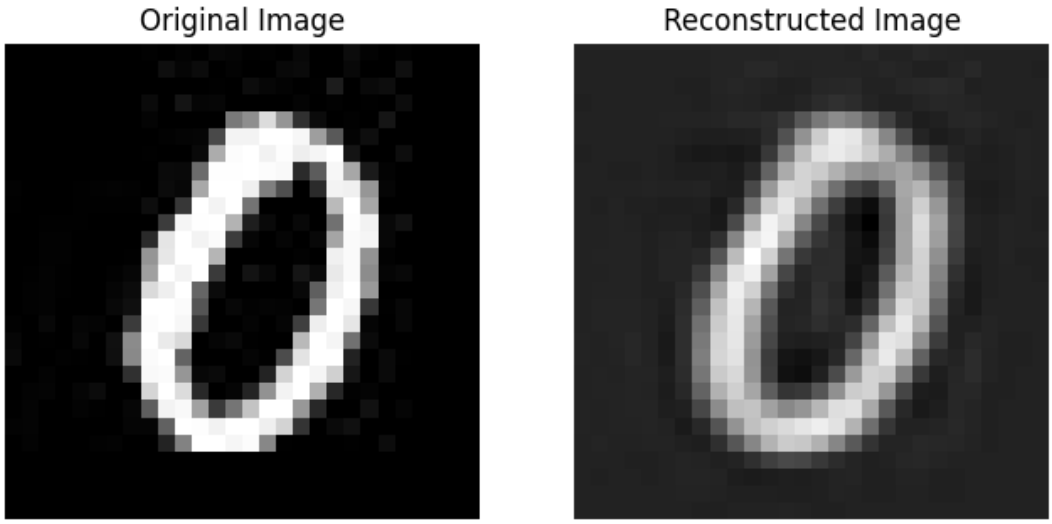

Figure 7: Comparison of Original and Reconstructed Images. The reconstructed image is obtained using PCA with $k = 128$ principal components.

As observed, the reconstructed image retains the major structural details of the original but lacks finer textures, which is a trade-off inherent in PCA-based compression. Increasing k would improve the reconstruction quality at the cost of higher computational complexity.

2.8 Confusion Matrix and Performance Metrics

2.8.1 Problem Statement

In this section, we analyze the performance of various multi-class classification models, namely:

- **Multilayer Perceptron (MLP)**
- **Convolutional Neural Network (CNN)**
- **MLP with PCA-based Features**
- **Multinomial Logistic Regression**

The models are evaluated using confusion matrices and four key classification metrics for each class:

1. **Accuracy**: The overall proportion of correct predictions.
2. **Precision**: The fraction of correctly predicted positive instances among all predicted positives.
3. **Recall**: The fraction of correctly predicted positive instances among all actual positives.
4. **F1 Score**: The harmonic mean of precision and recall.

2.8.2 Confusion Matrix Representation

For a multi-class classification problem with C classes, the confusion matrix is a $C \times C$ matrix, where the entry at row i and column j represents the number of instances of class i that were predicted as class j. The elements of the confusion matrix are used to compute the performance metrics for each class:

- **True Positives (TP)**: Number of times a class was correctly predicted.
- **False Positives (FP)**: Number of times a different class was incorrectly predicted as the target class.
- **False Negatives (FN)**: Number of times the target class was incorrectly predicted as a different class.
- **True Negatives (TN)**: Number of instances where neither the actual nor predicted class corresponds to the target class.

The performance metrics are computed as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{9}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{10}$$

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{11}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{12}$$

2.8.3 Visualization of Confusion Matrices

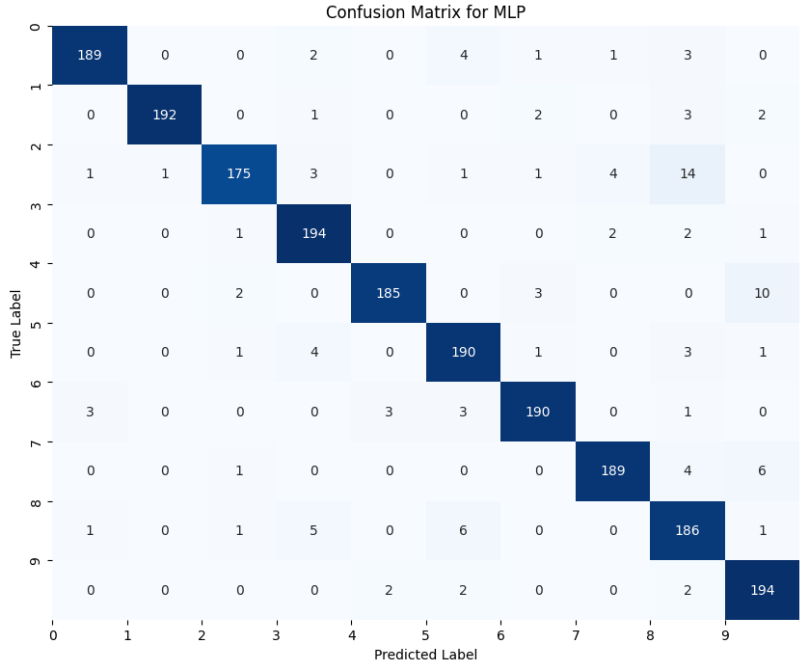

Figure 8: Confusion Matrix for MLP

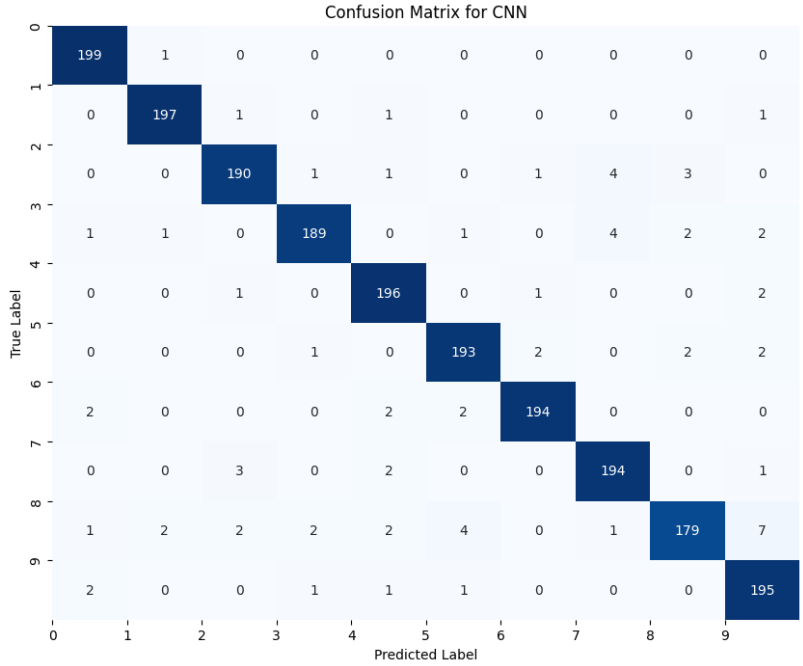

Figure 9: Confusion Matrix for CNN

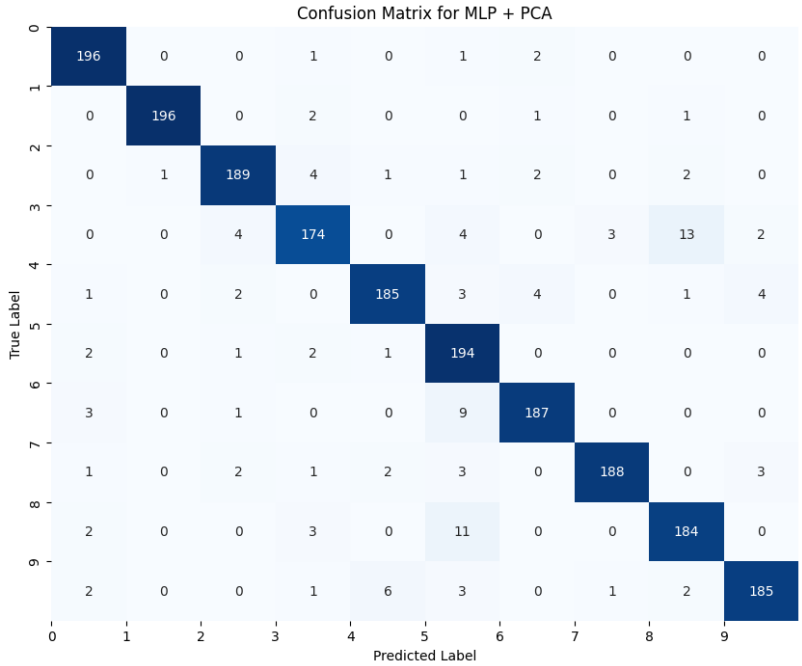

Figure 10: Confusion Matrix for MLP with PCA

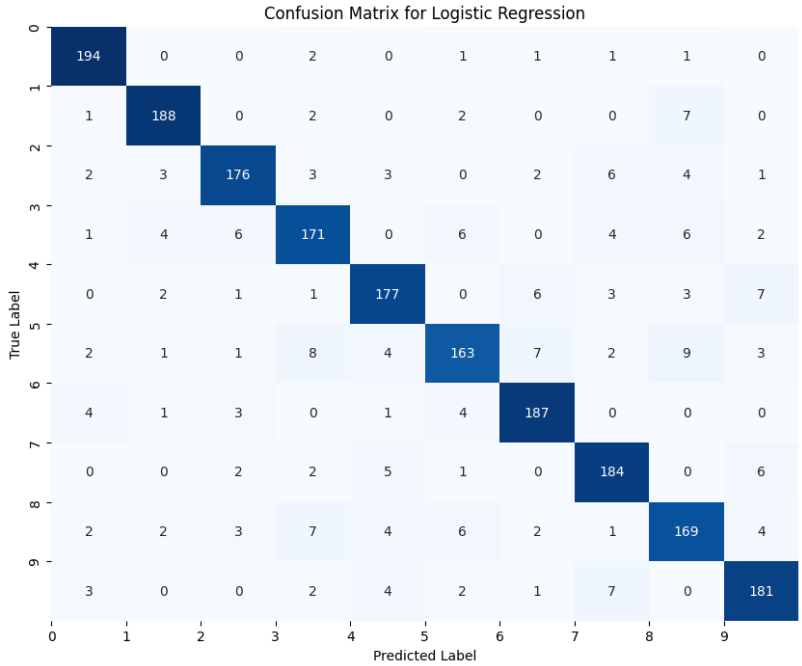

Figure 11: Confusion Matrix for Logistic Regression

2.8.4 Performance Metrics for Each Class

The following tables summarize the accuracy, precision, recall, and F1-score for each class across different models.

Table 1: Performance Metrics for Class 0

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
MLP	0.9920	0.974227	0.945	0.959391
CNN	0.9965	0.970732	0.995	0.982716
MLP + PCA	0.9925	0.946860	0.980	0.963145
Logistic Regression	0.9895	0.928230	0.970	0.948655

Table 2: Performance Metrics for Class 1

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
MLP	0.9955	0.994819	0.960	0.977099
CNN	0.9965	0.980100	0.985	0.982544
MLP + PCA	0.9975	0.994924	0.980	0.987406
Logistic Regression	0.9875	0.935323	0.940	0.937656

Table 3: Performance Metrics for Class 2

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
MLP	0.9845	0.966851	0.875	0.918635
CNN	0.9915	0.964467	0.950	0.957179
MLP + PCA	0.9895	0.949749	0.945	0.947368
Logistic Regression	0.9800	0.916667	0.880	0.897959

Table 4: Performance Metrics for Class 3

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
MLP	0.9895	0.928230	0.970	0.948655
CNN	0.9920	0.974227	0.945	0.959391
MLP + PCA	0.9800	0.925532	0.870	0.896907
Logistic Regression	0.9720	0.863636	0.855	0.859296

Table 5: Performance Metrics for Class 4

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
MLP	0.9900	0.973684	0.925	0.948718
CNN	0.9935	0.956098	0.980	0.967901
MLP + PCA	0.9875	0.948718	0.925	0.936709
Logistic Regression	0.9780	0.893939	0.885	0.889447

Table 6: Performance Metrics for Class 5

Model	Accuracy	Precision	Recall	F1 Score
MLP	0.9870	0.922330	0.950	0.935961
CNN	0.9925	0.960199	0.965	0.962594
MLP + PCA	0.9795	0.847162	0.970	0.904429
Logistic Regression	0.9705	0.881081	0.815	0.846753

Table 7: Performance Metrics for Class 6

Model	Accuracy	Precision	Recall	F1 Score
MLP	0.991	0.959596	0.950	0.954774
CNN	0.995	0.979798	0.970	0.974874
MLP + PCA	0.989	0.954082	0.935	0.944444
Logistic Regression	0.984	0.907767	0.935	0.921182

Table 8: Performance Metrics for Class 7

Model	Accuracy	Precision	Recall	F1 Score
MLP	0.9910	0.964286	0.945	0.954545
CNN	0.9925	0.955665	0.970	0.962779
MLP + PCA	0.9920	0.979167	0.940	0.959184
Logistic Regression	0.9800	0.884615	0.920	0.901961

Table 9: Performance Metrics for Class 8

Model	Accuracy	Precision	Recall	F1 Score
MLP	0.9770	0.853211	0.930	0.889952
CNN	0.9860	0.962366	0.895	0.927461
MLP + PCA	0.9825	0.906404	0.920	0.913151
Logistic Regression	0.9695	0.849246	0.845	0.847118

Table 10: Model Performance Ranking (Increasing Order)

Metric	Performance Order (Lowest to Highest)
Accuracy	Logistic Regression $\rightarrow$ MLP + PCA $\rightarrow$ MLP $\rightarrow$ CNN
Precision	Logistic Regression $\rightarrow$ MLP + PCA $\rightarrow$ MLP $\rightarrow$ CNN
Recall	Logistic Regression $\rightarrow$ MLP + PCA $\rightarrow$ MLP $\rightarrow$ CNN
F1 Score	Logistic Regression $\rightarrow$ MLP + PCA $\rightarrow$ MLP $\rightarrow$ CNN

2.8.5 Summary of Performance Trends

From the results, we observe the following:

- The CNN model consistently achieves the highest accuracy across most classes.
- MLP with PCA features shows a slight degradation in performance, indicating some loss of information due to dimensionality reduction.

- Logistic Regression generally performs worse than the other models, especially in terms of precision and F1-score.
- Certain classes (e.g., Class 8) have lower precision and recall across all models, possibly due to class imbalance or similarity to other digits.

2.9 Average AUC Score using ROC Curves

Problem Statement

Compute the average AUC score using ROC curves for each class obtained from the 10 binary classifiers (Logistic Regression) trained in Task 5. (1 mark)

Methodology

To evaluate the performance of the one-vs-rest (OvR) logistic regression classifiers, we compute the Receiver Operating Characteristic (ROC) curve for each class. The ROC curve is generated by varying the classification threshold and plotting the True Positive Rate (TPR) against the False Positive Rate (FPR). The Area Under the Curve (AUC) score is then computed to quantify the classifier's performance.

For each class, the following steps were performed:

- Compute model predictions for the validation dataset.
- Vary the classification threshold from 0 to 1 and compute TPR and FPR.
- Plot the ROC curve for each class.
- Compute the AUC score using numerical integration.
- Compute the average AUC score across all classes.

Results and Analysis

Figure **??** illustrates the ROC curves for each of the 10 classes, demonstrating the trade-off between sensitivity and specificity. The AUC scores for each class are reported in Table 11. A higher AUC score indicates better classification performance.

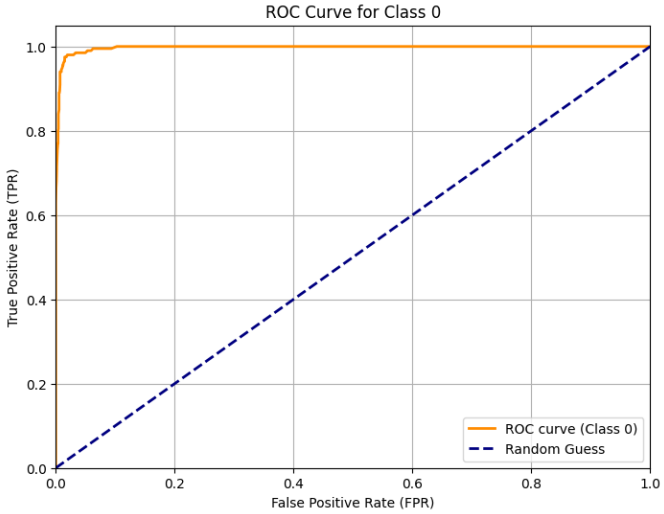

Figure 12: ROC Curve for Class 0

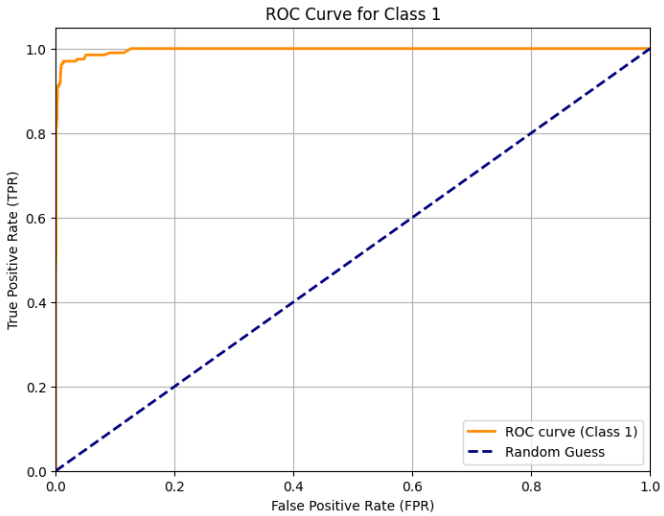

Figure 13: ROC Curve for Class 1

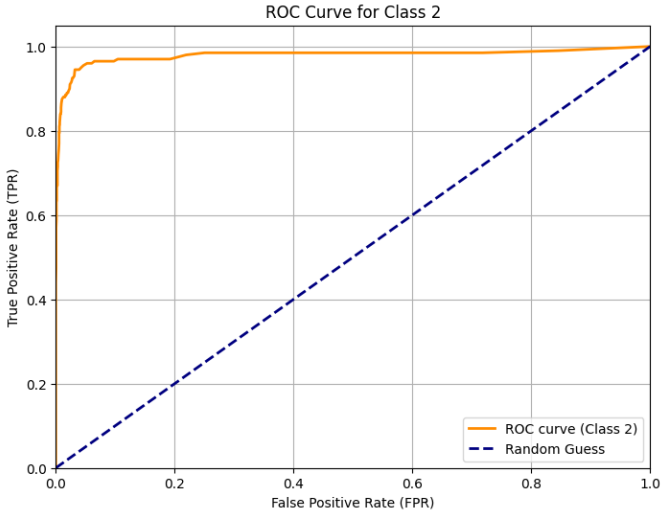

Figure 14: ROC Curve for Class 2

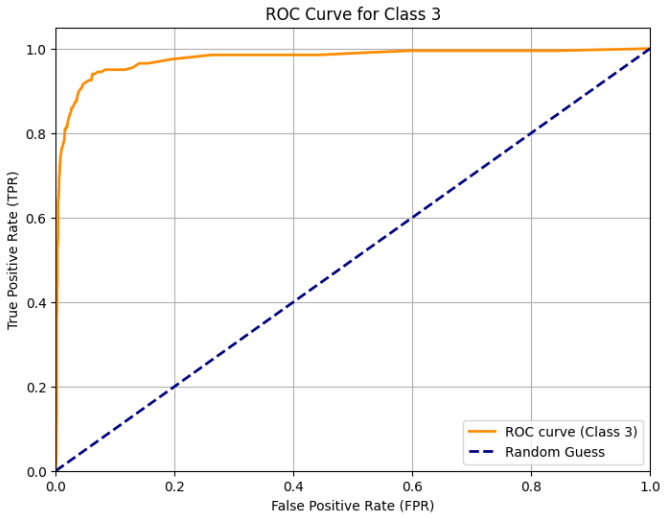

Figure 15: ROC Curve for Class 3

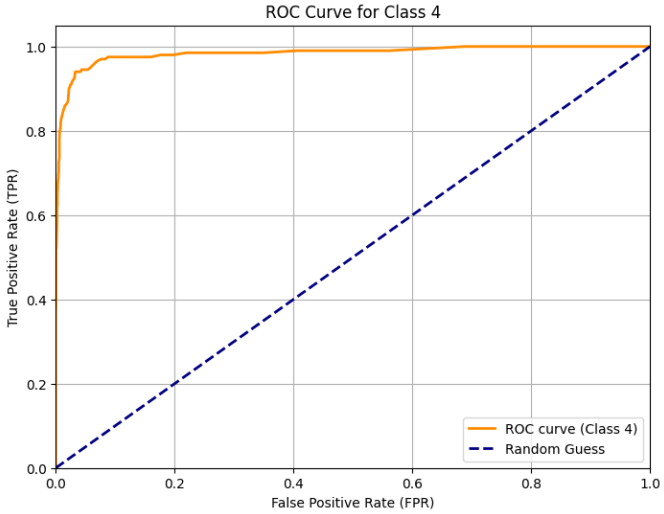

Figure 16: ROC Curve for Class 4

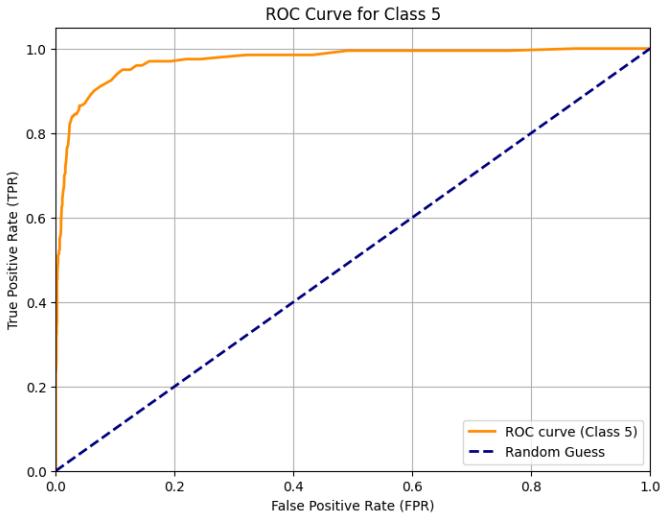

Figure 17: ROC Curve for Class 5

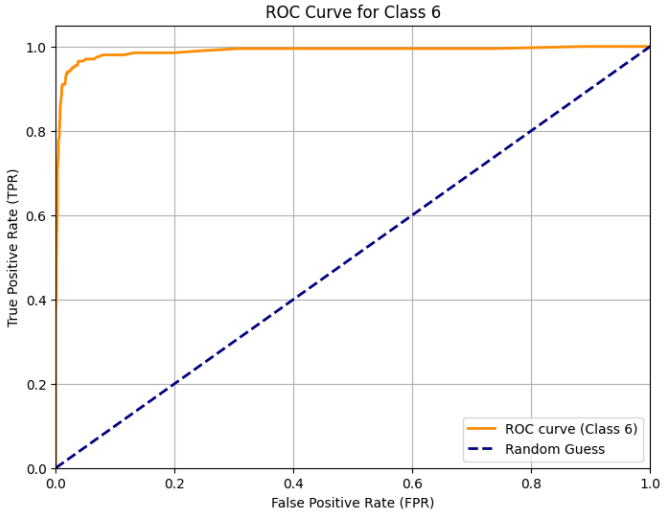

Figure 18: ROC Curve for Class 6

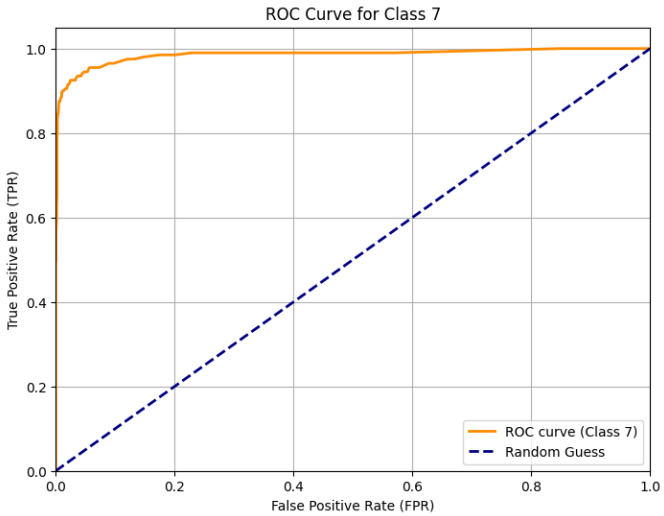

Figure 19: ROC Curve for Class 7

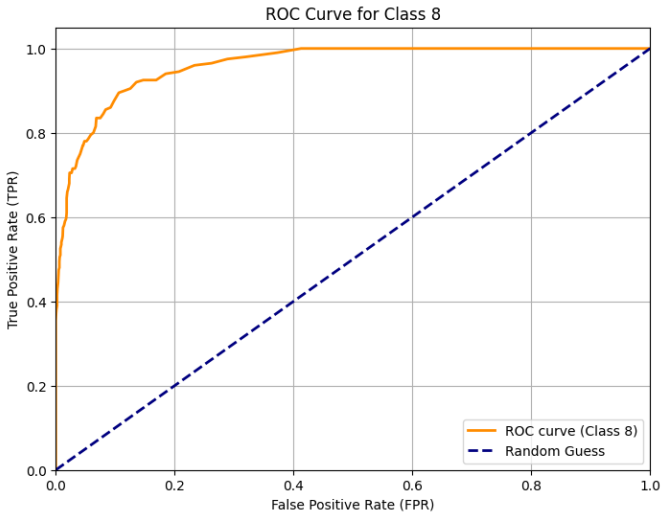

Figure 20: ROC Curve for Class 8

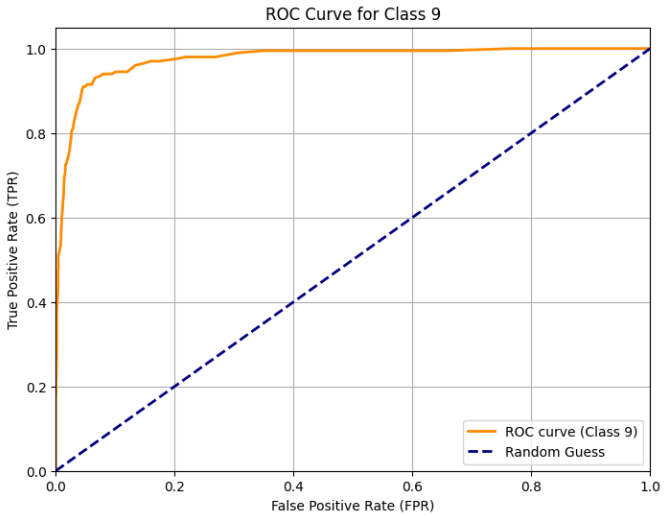

Figure 21: ROC Curve for Class 9

Table 11: AUC Scores for Each Class

Class	**AUC Score**
0	0.9971
1	0.9968
2	0.9791
3	0.9761
4	0.9840
5	0.9720
6	0.9887
7	0.9856
8	0.9608
9	0.9753

Conclusion

The results demonstrate that the logistic regression OvR classifiers achieve consistently high AUC scores across all classes, with an average AUC score of 0.9816. This indicates that the classifiers have strong discriminative power in distinguishing between classes. However, Class 8 has the lowest AUC score (0.9608), suggesting that it may be more challenging to separate from other classes, likely due to overlapping feature distributions.

3 Linear and Ridge Regression

3.1 Linear Regression

Linear regression aims to model the relationship between a dependent variable $Y \in \mathbb{R}^n$ and an independent variable matrix $X \in \mathbb{R}^{n \times d}$ by minimizing the sum of squared residuals. The optimization problem is formulated as follows:

$$\min_w J(w) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - w^T x^{(i)})^2 = \frac{1}{2n} \|Y - Xw\|^2. \tag{13}$$

Derivation of Ordinary Least Squares(OLS) solution

To obtain the optimal weight vector w that minimizes the cost function $J(w)$, we differentiate the objective function with respect to w and set it to zero:

$$\frac{\partial J(w)}{\partial w} = \frac{\partial}{\partial w} \left(\frac{1}{2n} \|Y - Xw\|^2 \right). \tag{14}$$

Expanding the norm term:

$$\|Y - Xw\|^2 = (Y - Xw)^T (Y - Xw). \tag{15}$$

Applying the derivative:

$$\frac{\partial}{\partial w} \left(\frac{1}{2n} (Y - Xw)^T (Y - Xw) \right) = -\frac{1}{n} X^T (Y - Xw). \tag{16}$$

Setting the derivative to zero:

$$X^T (Y - Xw) = 0. \tag{17}$$

Rearranging:

$$X^T X w = X^T Y. \tag{18}$$

Assuming $X^T X$ is invertible, the optimal solution for w is given by:

$$w = (X^T X)^{-1} X^T Y. \tag{19}$$

3.2 Ridge Regression

Ridge Regression is a regularized variant of Ordinary Least Squares (OLS) that introduces an L_2 penalty to prevent overfitting. Given a dataset with feature matrix $X \in \mathbb{R}^{n \times d}$ and response vector $Y \in \mathbb{R}^n$, the Ridge Regression optimization problem is formulated as:

$$\min_w J(w) = \frac{1}{2n}\|Y - Xw\|^2 + \frac{\lambda}{2}\|w\|^2.$$

Here:

- $X \in \mathbb{R}^{n \times d}$ represents n samples with d features each. - $Y \in \mathbb{R}^n$ is the vector of target values. - $w \in \mathbb{R}^d$ is the weight vector to be estimated. - $\lambda > 0$ is the regularization parameter that controls the trade-off between model complexity and fit to the training data.

Derivation of the Closed-Form Solution

To obtain the optimal w, we differentiate the objective function with respect to w and set the derivative to zero.
Expanding the first term:

$$\|Y - Xw\|^2 = (Y - Xw)^T(Y - Xw).$$

Taking the derivative:

$$\frac{\partial}{\partial w}\left[\frac{1}{2n}(Y - Xw)^T(Y - Xw)\right] = -\frac{1}{n}X^T(Y - Xw).$$

For the regularization term:

$$\frac{\partial}{\partial w}\left[\frac{\lambda}{2}\|w\|^2\right] = \lambda w.$$

Setting the derivative of the objective function to zero:

$$-\frac{1}{n}X^T(Y - Xw) + \lambda w = 0.$$

Rearranging:

$$X^TY = X^TXw + n\lambda w.$$

Factoring out w:

$$(X^TX + n\lambda I)w = X^TY.$$

Multiplying both sides by $(X^TX + n\lambda I)^{-1}$ (assuming it is invertible):

$$w = (X^TX + n\lambda I)^{-1}X^TY.$$

This closed-form solution is computationally efficient for small to moderate-sized datasets but may be numerically unstable if X^TX is ill-conditioned.

3.3 Deliverables

When OLS is not possible

If X does not have full column rank, meaning its columns are linearly dependent, then $X^T X$ is singular (non-invertible). This happens when:

- There are more features than samples ($d > n$), leading to a non-full-rank matrix.
- Some features are linear combinations of others, introducing redundancy.

Since $X^T X$ is singular, the standard OLS solution,

$$w_{\text{OLS}} = (X^T X)^{-1} X^T Y,$$

is not computable because $(X^T X)^{-1}$ does not exist.

Alternative Approaches

To handle this issue, we can:

- Use the **Moore-Penrose pseudoinverse**, which provides a least-norm solution:

  $$w_{\text{OLS}} = X^+ Y,$$

  where $X^+ = (X^T X)^+ X^T$ is the pseudoinverse.
- Apply **Ridge Regression**, which adds a regularization term λI to $X^T X$, ensuring invertibility:

  $$w_{\text{RR}} = (X^T X + \lambda I)^{-1} X^T Y.$$

Evaluation of Model Performance

The performance of the models is evaluated using the Mean Squared Error (MSE) on both the training and test datasets. The results are summarized below:

- Variance of Y: 0.3403
- OLS Train MSE: 0.0236
- Ridge Regression (RR) Train MSE: 0.0243
- OLS Test MSE: 0.0648
- Ridge Regression (RR) Test MSE: 0.0736

The learned weight vectors for both models are:

$$w_{\text{OLS}} = \begin{bmatrix} 0.3579 \\ 0.1762 \\ 0.0733 \\ 0.0975 \\ 0.3129 \end{bmatrix}, \quad w_{\text{RR}} = \begin{bmatrix} 0.3436 \\ 0.1710 \\ 0.0648 \\ 0.0994 \\ 0.2882 \end{bmatrix}$$

4 Support Vector Regression

4.1 Linear SVR Dual

Problem Statement

The goal is to solve the dual problem of the linear slack Support Vector Regression (SVR) using the ϵ-insensitive loss function. We employ the convex optimization library `cvxopt` to solve the quadratic programming (QP) formulation of the dual problem.

4.1.1 SVR Dual Formulation

Support Vector Regression (SVR) with the ϵ-insensitive loss function aims to fit a linear model:

$$f(x) = w^T x + b$$

such that deviations from the true target values y_i are penalized only if they exceed a given margin ϵ. Mathematically, the primal formulation is:

$$\min_{w,b,\xi_i,\xi_i^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

subject to the constraints:

$$y_i - w^T x_i - b \leq \epsilon + \xi_i, \quad \forall i$$

$$w^T x_i + b - y_i \leq \epsilon + \xi_i^*, \quad \forall i$$

$$\xi_i, \xi_i^* \geq 0, \quad \forall i.$$

Using Lagrange multipliers and deriving the dual, we obtain the following quadratic optimization problem:

$$\max_{\alpha,\alpha^*} \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)y_i - \epsilon\sum_{i=1}^{n}(\alpha_i^* + \alpha_i) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(x_i, x_j)$$

subject to the constraints:

$$\sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0, \quad 0 \leq \alpha_i, \alpha_i^* \leq C.$$

where $K(x_i, x_j) = x_i^T x_j$ for the linear case.

4.1.2 Mapping to `cvxopt` Quadratic Programming

The optimization problem in `cvxopt` is expressed in the form:

$$\min_x \frac{1}{2}x^T P x + q^T x$$

subject to:

$$Gx \leq h, \quad Ax = b.$$

For the SVR dual problem, the optimization is formulated as follows:

1. **Quadratic Term (P):** The matrix $P \in \mathbb{R}^{2n \times 2n}$ is constructed as:

$$P = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix}$$

   where $K \in \mathbb{R}^{n \times n}$ is the kernel matrix.

2. **Linear Term (q):** The vector $q \in \mathbb{R}^{2n \times 1}$ is given by:

$$q = \begin{bmatrix} \epsilon - y \\ \epsilon + y \end{bmatrix}.$$

   where $y \in \mathbb{R}^{n \times 1}$.

3. **Inequality Constraints (G and h):** The constraints $0 \leq \alpha_i, \alpha_i^* \leq C$ are represented as:

$$G \in \mathbb{R}^{4n \times 2n}, \quad G = \begin{bmatrix} -I_{2n} \\ I_{2n} \end{bmatrix}, \quad h \in \mathbb{R}^{4n \times 1}, \quad h = \begin{bmatrix} 0_{2n} \\ C_{2n} \end{bmatrix}.$$

4. **Equality Constraint (A and b):** The constraint $\sum(\alpha_i - \alpha_i^*) = 0$ is written as:

$$A \in \mathbb{R}^{1 \times 2n}, \quad A = \begin{bmatrix} 1_n & -1_n \end{bmatrix}, \quad b \in \mathbb{R}^{1 \times 1}, \quad b = 0.$$

4.1.3 Solution and Computation of Decision Variables

The solution vector $\alpha = [\alpha_1, \ldots, \alpha_n, \alpha_1^*, \ldots, \alpha_n^*]^T$ is obtained by solving the QP problem.

1. **Weight Vector (w):** The weight vector is computed as:

$$w = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) x_i.$$

2. **Bias Term (b):** The bias term is obtained using the support vectors:

$$b = y_i - w^T x_i, \quad \forall i \text{ in support vectors.}$$

3. **Slack Variables (ξ):** The slack variables are computed as:

$$\xi_i = \max(0, |y_i - w^T x_i - b| - \epsilon).$$

4.2 Graphical Results for SVR Predictions

To visualize the performance of the trained SVR models, we plot the predictions on the test set. Each plot includes:

- The predicted closing price.
- The actual closing price.
- The average price over the past t days.

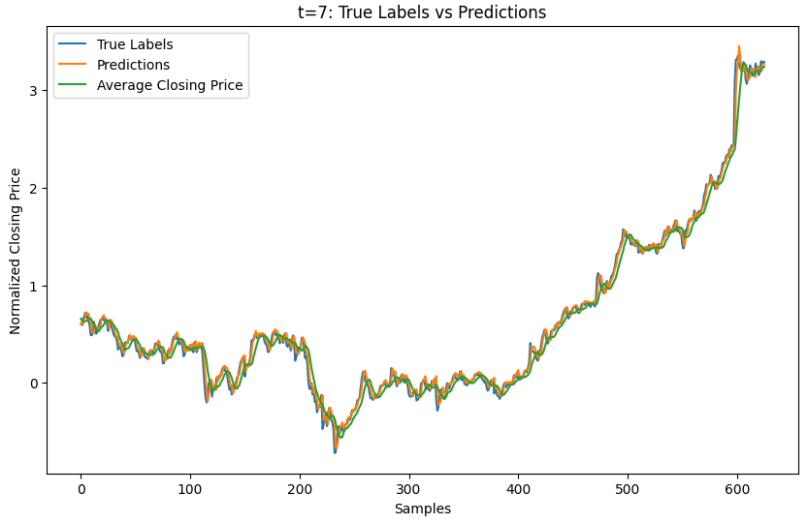

Figure 22: SVR Predictions for $t = 7$ (Linear)

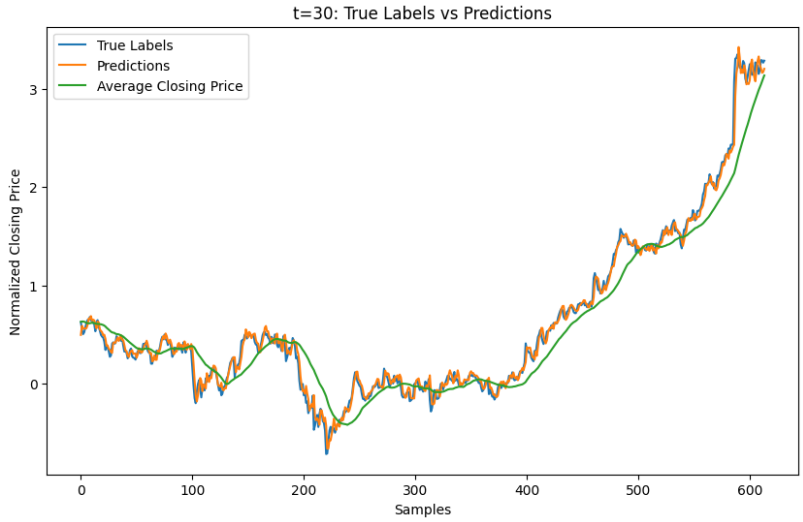

Figure 23: SVR Predictions for $t = 30$ (Linear)

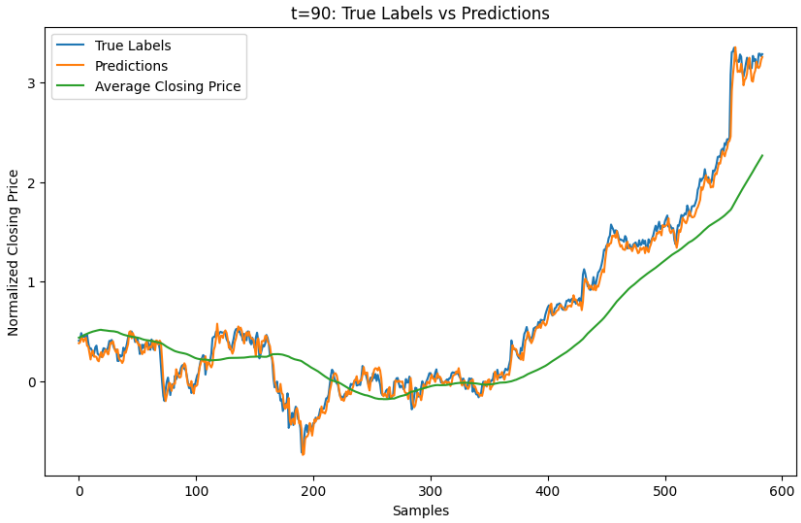

Figure 24: SVR Predictions for $t = 90$ (Linear)

5 Support Vector Regression with Gaussian Kernel

5.1 Gaussian SVR Dual

The dual formulation of SVR with the Radial Basis Function (RBF) kernel introduces Lagrange multipliers α_i and α_i^*, which are optimized under the ϵ-insensitive loss function.

5.1.1 Dual Formulation

The primal form of SVR is given by:

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

subject to the constraints:

$$y_i - w^T x_i - b \leq \epsilon + \xi_i, \quad \forall i$$

$$w^T x_i + b - y_i \leq \epsilon + \xi_i^*, \quad \forall i$$

$$\xi_i, \xi_i^* \geq 0, \quad \forall i.$$

Using the kernel trick, the dual problem is formulated as:

$$\max_{\alpha,\alpha^*} -\frac{1}{2}(\alpha - \alpha^*)^T K(\alpha - \alpha^*) + (\alpha - \alpha^*)^T y - \epsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*)$$

subject to:

$$\sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0, \quad 0 \leq \alpha_i, \alpha_i^* \leq C.$$

where K is the kernel matrix defined as:

$$K_{ij} = \exp\left(-\gamma \|x_i - x_j\|^2\right).$$

5.1.2 CVXOPT Input Mapping

CVXOPT solves quadratic programs of the form:

$$\min_x \frac{1}{2} x^T P x + q^T x$$

subject to:

$$Gx \leq h, \quad Ax = b.$$

For SVR:

1. **Quadratic Term (P):** The matrix $P \in \mathbb{R}^{2n \times 2n}$ is constructed as:

$$P = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix}$$

   where $K \in \mathbb{R}^{n \times n}$ is the kernel matrix.

2. **Linear Term (q):** The vector $q \in \mathbb{R}^{2n \times 1}$ is given by:

$$q = \begin{bmatrix} \epsilon - y \\ \epsilon + y \end{bmatrix}.$$

   Here, $y \in \mathbb{R}^{n \times 1}$ is the target vector.

3. **Inequality Constraints (G, h):** The constraints $0 \leq \alpha_i, \alpha_i^* \leq C$ are represented as:

$$G = \begin{bmatrix} -I_{2n} \\ I_{2n} \end{bmatrix} \in \mathbb{R}^{4n \times 2n}, \quad h = \begin{bmatrix} 0_{2n} \\ C_{2n} \end{bmatrix} \in \mathbb{R}^{4n \times 1}.$$

   Here, I_{2n} is the $2n \times 2n$ identity matrix.

4. **Equality Constraint (A, b):** The constraint $\sum(\alpha_i - \alpha_i^*) = 0$ is written as:

$$A = \begin{bmatrix} 1_n & -1_n \end{bmatrix} \in \mathbb{R}^{1 \times 2n}, \quad b = 0 \in \mathbb{R}^{1 \times 1}.$$

The solution vector α contains α_i and α_i^*, which define the optimal decision function.

5.2 Hyperparameter Tuning

We perform a grid search over a set of values for C and ϵ to determine the best combination. The search space is:

- $C \in \{0.01, 0.1, 1, 10, 100\}$
- $\epsilon \in \{0.001, 0.01, 0.1, 0.5, 1\}$

For each combination, we solve the dual problem using CVXOPT and evaluate performance using the ϵ-insensitive loss. The best hyperparameters are selected based on the lowest test set error.

Results:

- Best C: 100
- Best ϵ: 0.01
- Best ϵ-insensitive loss (on $t = 90$): 0.27126
- Test Loss for $t = 7$: 0.15293
- Test Loss for $t = 30$: 0.19642

Since tuning for $t = 90$ yielded the lowest test loss across all t, we use this C and ϵ for all subsequent evaluations.

5.3 Plotting SVR Predictions

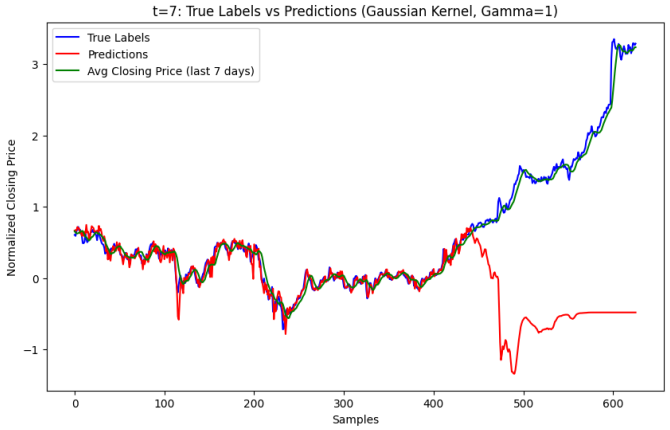

Figure 25: Gaussian SVR Predictions for $\gamma = 1, t = 7$

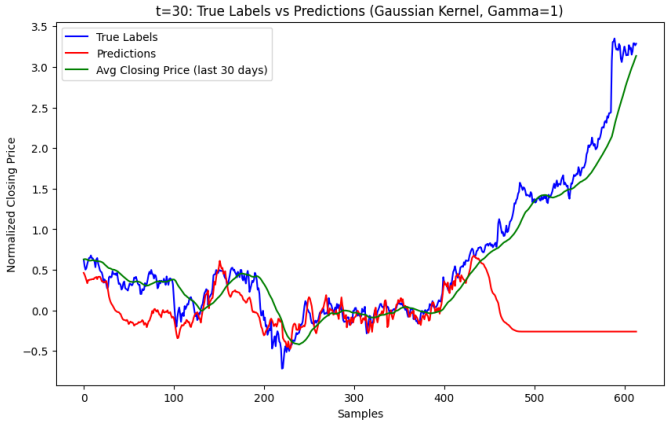

Figure 26: Gaussian SVR Predictions for $\gamma = 1, t = 30$

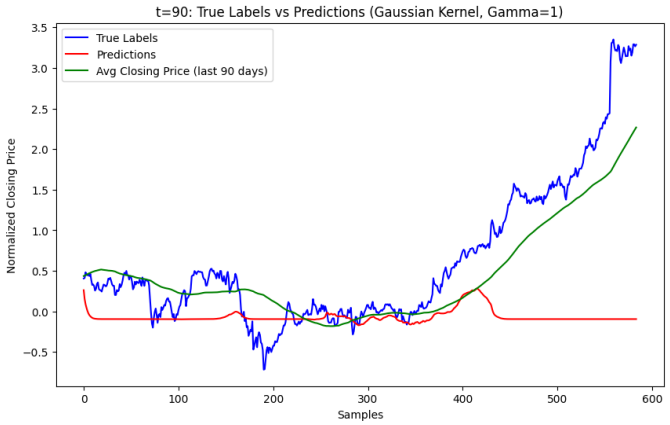

Figure 27: Gaussian SVR Predictions for $\gamma = 1, t = 90$

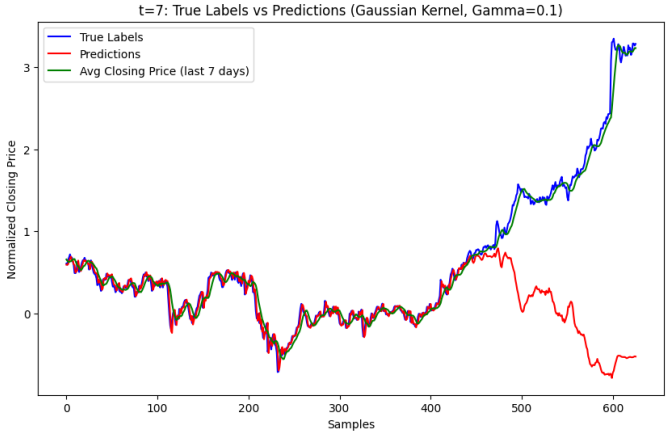

Figure 28: Gaussian SVR Predictions for $\gamma = 0.1, t = 7$

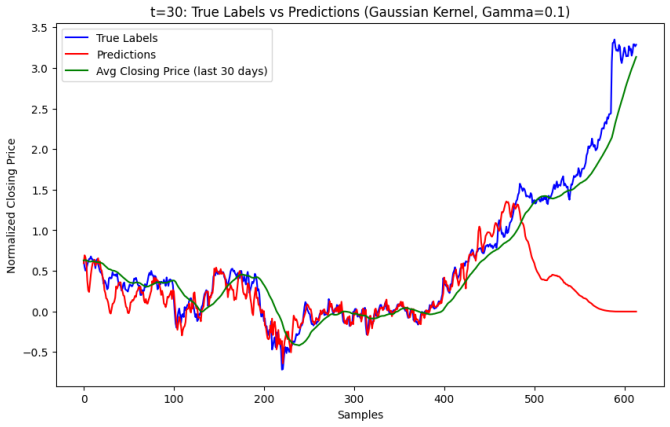

Figure 29: Gaussian SVR Predictions for $\gamma = 0.1, t = 30$

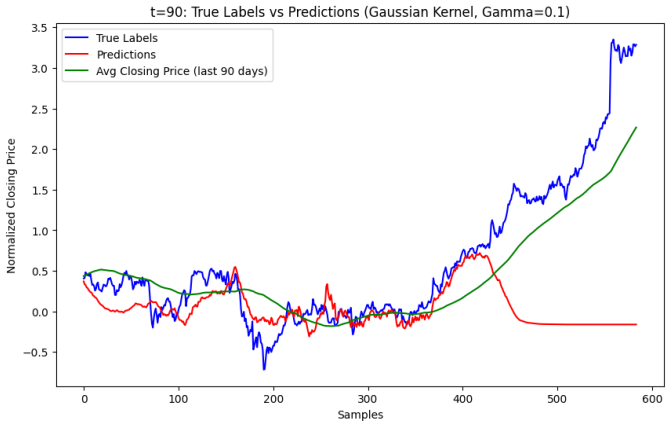

Figure 30: Gaussian SVR Predictions for $\gamma = 0.1, t = 90$

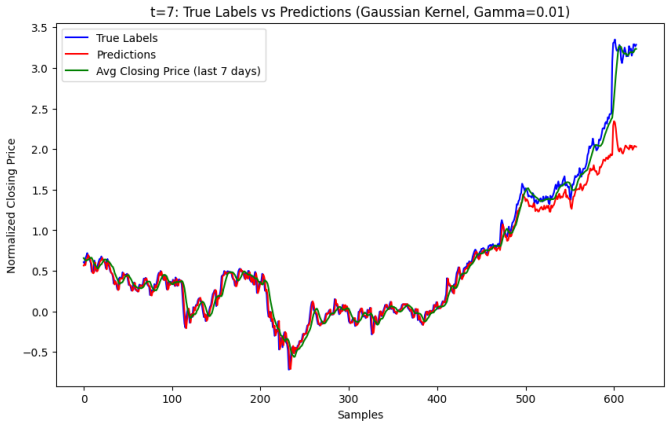

Figure 31: Gaussian SVR Predictions for $\gamma = 0.01, t = 7$

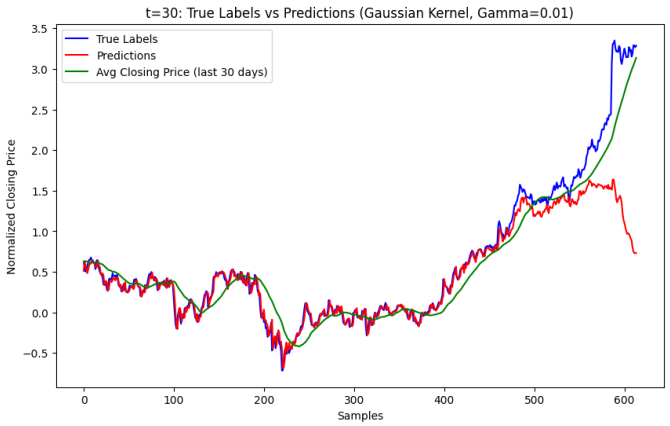

Figure 32: Gaussian SVR Predictions for $\gamma = 0.01, t = 30$

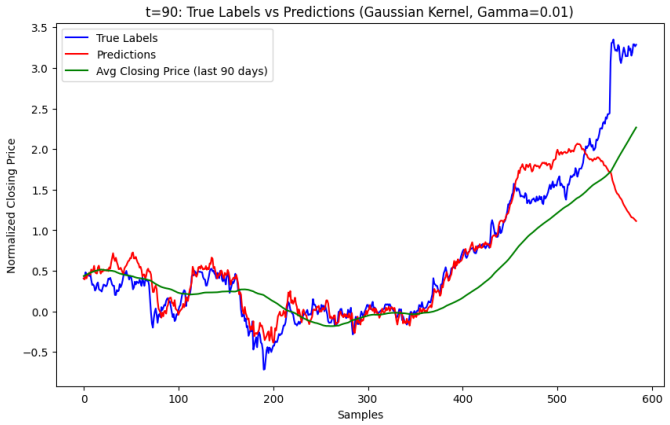

Figure 33: Gaussian SVR Predictions for $\gamma = 0.01, t = 90$

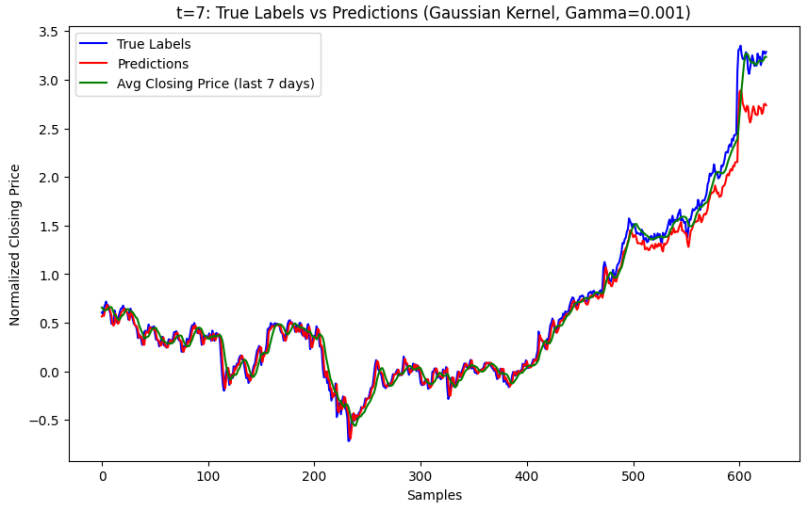

Figure 34: Gaussian SVR Predictions for $\gamma = 0.001, t = 7$

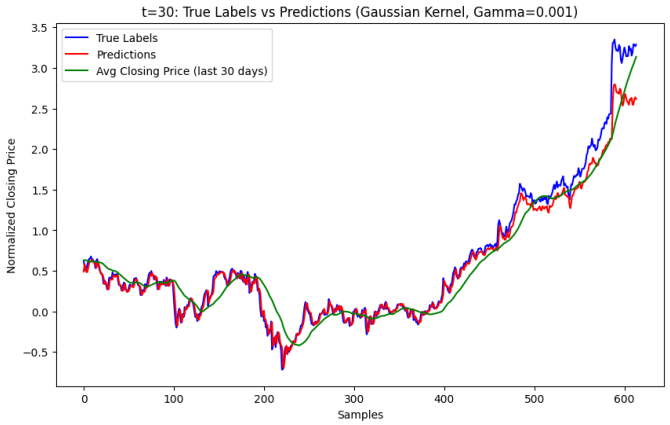

Figure 35: Gaussian SVR Predictions for $\gamma = 0.001, t = 30$

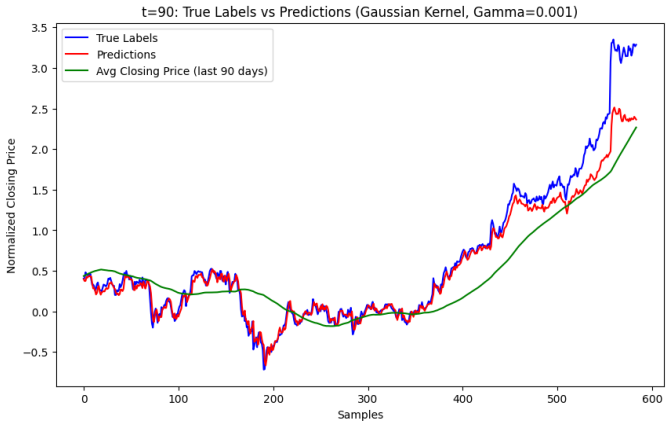

Figure 36: Gaussian SVR Predictions for $\gamma = 0.001, t = 90$