AIML_2025_A2_23634

KOLIPAKA BHARGAV SASHANK
23634

March 28, 2025

1 Support Vector Machine and Perceptron

1.1 Introduction

This report investigates the separability of the CIFAR-100 dataset by applying the Perceptron algorithm and Support Vector Machines (SVMs). The goal is to determine if the dataset is separable and analyze the sources of non-separability.

1.2 Tasks and Implementation

1.2.1 Perceptron Algorithm

The perceptron algorithm was implemented and run on a subset of CIFAR-100 dataset. The convergence was examined based on the misclassification rate over iterations. The following plots given in the last pages show the misclassification rate versus the number of iterations.

The maximum number of iterations was first set to 10000, then it was increased to 50000, then to 100000, then to 500000 and finally to 1000000. It can be seen in the plot that the misclassification rate is oscillating between fixed values of misclassification rate (*between 0.3 and 0.5*) and is not going behind or beyond those values which means that the perceptron algorithm is not convergent on the dataset given.

1.2.2 Linear SVM

A soft-margin SVM with a linear kernel was trained using both the primal and dual formulations, solved using the `solvers.qp` function of the `cvxopt` package. The results were compared in terms of runtime.

The primal SVM formulation is given by:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \xi_i \tag{1}$$

subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i. \tag{2}$$

The dual SVM formulation is given by:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{3}$$

subject to:

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i. \tag{4}$$

1.2.3 Kernelized SVM

A Gaussian kernel SVM was implemented to address non-separability. Hyperparameters C, γ were tuned to ensure the decision boundary effectively classified the data and we achieved 0 misclassifications for C = 10 and γ = 100.

1.2.4 Retraining the Perceptron

After isolating the non-separable images using the linear SVM, those images were removed, and the perceptron was retrained to check if it converged and it is converging at 1185th iteration.

1.3 Results and Analysis

1.3.1 Perceptron Misclassification Rate (Task 1)

The first five figures below show the relationship between misclassification rate and number of iterations. The plots show that the perceptron algorithm is not converging.

1.3.2 Comparison of Primal and Dual SVM Solutions (Task 2)

The primal and dual SVMs were run separately, and the following observations were noted: - Primal SVM runtime: 1.1613194942474365 seconds - Dual SVM runtime: 1.115084171295166 seconds

The dual formulation was found to be *faster* than that of the primal.

1.3.3 Images Causing Non-Separability (Task 3)

The indices of the images identified as causing non-separability are in the CSV file named inseparable_23634.csv. It consists of 403 images which cause non-separability of the dataset given.

1.3.4 Final Misclassification Rate of Kernelized SVM (Task 4)

The misclassification rate of the kernelized SVM was found to be 0% for C = 10 and $\gamma = 100$.

1.3.5 Retrained Perceptron Misclassification Rate (Task 5)

The perceptron was retrained after removing non-separable images. The last figure(plot) shows the misclassification rate over the number of iterations and it converges at 1185th iteration.

This confirms that the perceptron converges after removing the non-separable samples.

1.4 Conclusion

This case study verified that the perceptron does not converge on a non-separable dataset. A linear SVM was used to isolate non-separable samples, and a kernelized SVM demonstrated improved classification by scaling the . After removing non-separable images, the perceptron successfully converged, confirming the role of those images in non-separability.

2 Logistic Regression, MLP, CNN, and PCA Report

2.1 Introduction

This report examines classification models using Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN), Principal Component Analysis (PCA), and Logistic Regression on a subset of the MNIST-JPG dataset. The performance of these models is compared based on accuracy, precision, recall, F1-score, and AUC scores.

2.2 Tasks and Implementation

2.2.1 Multi-Layer Perceptron (MLP)

An MLP model was trained on flattened images (28x28 = 784 dimensions). The architecture consisted of:
- Input layer: 784 neurons
- Hidden layers: 512, 256, 128 neurons (ReLU activation)
- Output layer: 10 neurons (Softmax activation)
The optimiser used is Stochastic Gradient Descent(SGD) equipped with momentum = 0.9 and learning rate = 0.01 and the criterion is Cross Entropy Loss

2.2.2 Convolutional Neural Network (CNN)

A CNN model was implemented to process raw images without flattening. The architecture consists of:
- Two convolutional layers with 3x3 kernels, the first having 32 filters and the second 64 filters.
- Max pooling layers with a 2x2 filter to downsample feature maps.
- A fully connected layer with 128 neurons, followed by the output layer with 10 neurons for classification.
- ReLU activation is used in all layers except the final one, which applies softmax.
The optimiser used is Stochastic Gradient Descent(SGD) equipped with momentum = 0.9 and learning rate = 0.01 and the criterion is Cross Entropy Loss

2.2.3 Principal Component Analysis (PCA)

PCA was applied to reduce the dimensionality of images before training classifiers.

Mathematically, PCA transforms the data matrix X as follows:

$$X' = XW_k \tag{5}$$

where W_k contains the top k eigenvectors of the covariance matrix $X^T X$ and k = 117 in this case.

2.2.4 MLP with PCA

The MLP model was trained again using PCA-reduced features consisting of :-

- Input layer: 117 neurons
- Hidden layers: 512, 256, 128 neurons (ReLU activation)
- Output layer: 10 neurons (Softmax activation)
The optimiser used is Stochastic Gradient Descent(SGD) equipped with momentum = 0.9 and learning rate = 0.01 and the criterion is Cross Entropy Loss

2.2.5 Logistic Regression with PCA

A logistic regression model was trained using PCA features with a one-vs-rest approach and a logistic regression model for multi-class classification.

2.3 Results and Analysis

2.3.1 Reconstructed Image from PCA (Task 1)

A sample image of each class reconstructed from principal components using PCA.

2.3.2 Confusion Matrices (Task 2)

The confusion matrices for all models were computed and compared using accuracy, precision, recall, and F1-score.

Confusion Matrix for MLP :-

$$\begin{bmatrix} 164 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3 & 0 \\ 0 & 203 & 1 & 1 & 0 & 0 & 2 & 2 & 1 & 1 \\ 3 & 1 & 207 & 1 & 1 & 0 & 2 & 2 & 2 & 0 \\ 0 & 0 & 0 & 170 & 0 & 2 & 0 & 1 & 5 & 2 \\ 0 & 0 & 0 & 0 & 199 & 0 & 0 & 0 & 2 & 8 \\ 3 & 1 & 0 & 2 & 0 & 206 & 1 & 1 & 0 & 0 \\ 3 & 0 & 2 & 0 & 0 & 3 & 201 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 2 & 0 & 0 & 178 & 0 & 7 \\ 0 & 2 & 1 & 1 & 0 & 4 & 1 & 1 & 207 & 1 \\ 0 & 1 & 0 & 3 & 4 & 1 & 0 & 1 & 1 & 170 \end{bmatrix}$$

Class	Precision	Recall	F1-score	Support
0	0.9425	0.9762	0.9591	168
1	0.9760	0.9621	0.9690	211
2	0.9718	0.9452	0.9583	219
3	0.9551	0.9444	0.9497	180
4	0.9660	0.9522	0.9590	209
5	0.9493	0.9626	0.9559	214
6	0.9710	0.9571	0.9640	210
7	0.9519	0.9368	0.9443	190
8	0.9367	0.9495	0.9431	218
9	0.8995	0.9392	0.9189	181
Accuracy			0.9525	2000
Macro avg	0.9520	0.9525	0.9521	2000
Weighted avg	0.9529	0.9525	0.9526	2000

Table 1: Classification Report for MLP

Confusion Matrix for CNN :-

$$\begin{bmatrix} 166 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 208 & 1 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 2 & 0 & 213 & 1 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 177 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 206 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 2 & 0 & 212 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 207 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 3 & 0 & 0 & 182 & 0 & 2 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 213 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 3 & 176 \end{bmatrix}$$

Confusion Matrix for MLP with PCA :-

Class	Precision	Recall	F1-score	Support
0	0.9651	0.9881	0.9765	168
1	0.9952	0.9858	0.9905	211
2	0.9861	0.9726	0.9793	219
3	0.9779	0.9833	0.9806	180
4	0.9763	0.9856	0.9810	209
5	0.9907	0.9907	0.9907	214
6	0.9952	0.9857	0.9904	210
7	0.9785	0.9579	0.9681	190
8	0.9595	0.9771	0.9682	218
9	0.9724	0.9724	0.9724	181
Accuracy		0.9800		2000
Macro Avg	0.9797	0.9799	0.9798	2000
Weighted Avg	0.9801	0.9800	0.9800	2000

Table 2: Classification Report for CNN

$$\begin{bmatrix} 166 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 206 & 1 & 0 & 0 & 0 & 2 & 1 & 0 & 1 \\ 2 & 0 & 207 & 3 & 4 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 170 & 0 & 1 & 0 & 1 & 4 & 3 \\ 0 & 1 & 1 & 0 & 194 & 0 & 0 & 0 & 3 & 10 \\ 1 & 1 & 0 & 5 & 0 & 203 & 1 & 1 & 2 & 0 \\ 2 & 0 & 2 & 0 & 1 & 2 & 202 & 0 & 0 & 1 \\ 1 & 0 & 2 & 1 & 4 & 0 & 0 & 178 & 0 & 4 \\ 0 & 1 & 0 & 4 & 1 & 2 & 4 & 2 & 202 & 2 \\ 0 & 1 & 0 & 2 & 7 & 2 & 0 & 3 & 1 & 165 \end{bmatrix}$$

Class	Precision	Recall	F1-score	Support
0	0.9651	0.9881	0.9765	168
1	0.9810	0.9763	0.9786	211
2	0.9673	0.9452	0.9561	219
3	0.9189	0.9444	0.9315	180
4	0.9194	0.9282	0.9238	209
5	0.9667	0.9486	0.9575	214
6	0.9573	0.9619	0.9596	210
7	0.9468	0.9368	0.9418	190
8	0.9484	0.9266	0.9374	218
9	0.8871	0.9116	0.8992	181
Accuracy	-	-	0.9465	2000
Macro Avg	0.9458	0.9468	0.9462	2000
Weighted Avg	0.9468	0.9465	0.9466	2000

Table 3: Classification Report for MLP with PCA

Confusion Matrix for Logistic Regression :-

$$\begin{bmatrix} 164 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 2 & 0 \\ 0 & 206 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 2 & 6 & 190 & 7 & 2 & 1 & 3 & 1 & 5 & 2 \\ 1 & 2 & 2 & 155 & 0 & 6 & 1 & 4 & 7 & 2 \\ 0 & 1 & 2 & 1 & 187 & 1 & 0 & 1 & 3 & 13 \\ 4 & 2 & 1 & 12 & 5 & 183 & 0 & 0 & 7 & 0 \\ 5 & 0 & 4 & 0 & 0 & 3 & 194 & 1 & 2 & 1 \\ 1 & 0 & 2 & 1 & 4 & 1 & 0 & 172 & 0 & 9 \\ 1 & 5 & 2 & 11 & 2 & 11 & 2 & 1 & 179 & 4 \\ 0 & 2 & 0 & 4 & 11 & 0 & 1 & 8 & 4 & 151 \end{bmatrix}$$

2.3.3 AUC Score and ROC Curve (Task 3)

The average AUC score was computed for the logistic regression classifiers trained with PCA features.

2.4 Conclusion

This study compared classification performance using MLP, CNN, and PCA-based approaches. PCA reduced dimensionality while maintaining classifica-

Class	Precision	Recall	F1-Score	Support
0	0.9213	0.9762	0.9480	168
1	0.9196	0.9763	0.9471	211
2	0.9314	0.8676	0.8983	219
3	0.8115	0.8611	0.8356	180
4	0.8779	0.8947	0.8863	209
5	0.8883	0.8551	0.8714	214
6	0.9604	0.9238	0.9417	210
7	0.9101	0.9053	0.9077	190
8	0.8524	0.8211	0.8364	218
9	0.8251	0.8343	0.8297	181
Accuracy		0.8905		2000
Macro Avg	0.8898	0.8915	0.8902	2000
Weighted Avg	0.8911	0.8905	0.8903	2000

Table 4: Classification Report for Logistic Regression

tion performance. The CNN performed best, followed by MLP, and logistic regression showed reasonable results with PCA features.

3 3.1 Linear Regression and Ridge Regression Report

3.1 Introduction

This report presents the implementation of two regression techniques: Ordinary Least Squares (OLS) and Ridge Regression (RR). The goal is to minimize the respective objective functions and analyze their performance on given datasets.

3.2 Mathematical Formulation

3.2.1 Ordinary Least Squares (OLS)

OLS aims to minimize the squared error loss function:

$$J(w) = \frac{1}{2n}\sum_{i=1}^{n}(y^{(i)} - w^T x^{(i)})^2 = \frac{1}{2n}||Y - Xw||^2, \tag{6}$$

where $X \in R^{n \times d}$ represents the feature matrix, $Y \in R^n$ is the target vector, and $w \in R^d$ is the weight vector.

To find the optimal weight vector, we compute the gradient:

$$\nabla J(w) = -\frac{1}{n} X^T (Y - Xw) = 0. \tag{7}$$

Solving for w, we obtain the closed-form solution (assuming that $X^T X$ is invertible):

$$w^{\text{ols}} = (X^T X)^{-1} X^T Y. \tag{8}$$

3.2.2 Ridge Regression (RR)

Ridge regression introduces a regularization term to prevent overfitting by minimizing:

$$J(w) = \frac{1}{2n} ||Y - Xw||^2 + \frac{\lambda}{2} ||w||^2. \tag{9}$$

Taking the gradient and setting it to zero:

$$-\frac{1}{n} X^T (Y - Xw) + \lambda w = 0. \tag{10}$$

Solving for w, we obtain:

$$w^{\text{rr}} = (X^T X + \lambda I)^{-1} X^T Y. \tag{11}$$

3.3 Analysis and Discussion

3.3.1 Can OLS be performed if X does not have full column rank?

No, if X does not have full column rank, then $X^T X$ is singular (non-invertible), making the OLS solution undefined. In such cases, Ridge Regression remains a viable alternative due to the added regularization term λI, which ensures invertibility. Lasso is also a good alternative.

3.3.2 Mean Squared Error (MSE) Calculation

The Mean Squared Error (MSE) is defined as:

$$MSE(w) = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - w^T x^{(i)})^2 = \frac{1}{n} ||Y - Xw||^2. \tag{12}$$

We computed the MSE for both w^{ols} and w^{rr} using the training datasets D_1^{train} and D_2^{train}.

MSE for $w^{\mathbf{ols}}$ on $D_1^{\mathbf{train}}$ $= 0.03567052046883622$
MSE for $w^{\mathbf{ols}}$ on $D_1^{\mathbf{test}}$ $= 0.09228164066035446$

$$\textbf{MSE for } w^{\mathbf{ols}} \textbf{ on } D_2^{\mathbf{train}} = 5.844188675406888 \times 10^{-29}$$

MSE for $w^{\mathbf{ols}}$ on $D_2^{\mathbf{test}}$ $= 29.374555476640488$
MSE for $w^{\mathbf{rr}}$ on $D_1^{\mathbf{train}}$ $= 0.036543631755336814$
MSE for $w^{\mathbf{rr}}$ on $D_1^{\mathbf{test}}$ $= 0.09252099641023462$
MSE for $w^{\mathbf{rr}}$ on $D_2^{\mathbf{train}}$ $= 0.003954081874105127$
MSE for $w^{\mathbf{rr}}$ on $D_2^{\mathbf{test}}$ $= 29.55734316603495$

3.3.3 Weight Vectors

The computed weight vectors, w^{ols}, w^{rr} for D_2^{train} are stored as CSV files:

- Ordinary Least Squares: `w_ols_23634.csv`
- Ridge Regression: `w_rr_23634.csv`

The weight vectors, w^{ols}, w^{rr} for D_1^{train} are

$$w^{\text{ols}} = \begin{bmatrix} 0.40266306 \\ 0.15955885 \\ 0.16895094 \\ 0.1729866 \\ 0.16841007 \end{bmatrix}$$

and

$$w^{\text{rr}} = \begin{bmatrix} 0.36496994 \\ 0.15506728 \\ 0.15628042 \\ 0.15457552 \\ 0.17858549 \end{bmatrix}$$

3.4 Conclusion

This report presents the mathematical formulations and implementations of Ordinary Least Squares and Ridge Regression. We derived the optimal weight vectors for both approaches and analyzed their performance based on Mean Squared Error.

4 Support Vector Regression Report

4.1 Problem Statement

In this report, we apply Support Vector Regression (SVR) to predict stock prices using historical stock market data. The goal is to train SVR models using different time window sizes $t \in \{7, 30, 90\}$ and evaluate their performance on test data.

4.1.1 Support Vector Regression Formulation

SVR minimizes a regularized error function given by:

$$\min_{w,b} C \sum_{n=1}^{N} E_\epsilon(f(x_n) - y_n) + \frac{1}{2}\|w\|^2, \tag{13}$$

where the ϵ-insensitive loss function is defined as:

$$E_\epsilon(f(x) - y) = \begin{cases} 0, & \text{if } |f(x) - y| < \epsilon, \\ |f(x) - y| - \epsilon, & \text{otherwise.} \end{cases} \tag{14}$$

The function $f(x)$ is represented as:

$$f(x) = w^T \phi(x) + b, \tag{15}$$

where $\phi(x)$ is a fixed feature space transformation.

4.2 Methodology

4.2.1 Data Preprocessing

The data processing steps are as follows:

1. Query the oracle to find the assigned stock ticker, "BAC" was returned
2. Download BAC.csv file from `stocknet-dataset/price/raw/`.
3. Extract closing prices and normalize them using:
$$x' = \frac{x - \mu}{\sigma}. \tag{16}$$
(used Scikit Learn's `StandardScaler`)
4. Construct an $(N - t) \times t$ matrix where each row contains closing prices of t previous days. This forms the feature matrix X.
5. Define labels Y by removing the first t elements of the price vector.
6. Split the dataset into training (50%) and test (50%) sets.

4.2.2 Training SVR Models

We train the following SVR models using the training set:

1. Solve the dual of the slack linear SVR using `cvxopt`.
2. Solve the dual of the kernelized SVR with an RBF kernel using $\gamma \in \{1, 0.1, 0.01, 0.001\}$, also using `cvxopt`.

4.3 Deliverables

For each trained SVR model, we generate graphs plotting:

- Predicted closing price values.
- Actual closing price values.
- Average closing price over the previous t days.

The plots are given below.

4.4 Conclusion

This report presents the implementation and evaluation of SVR for stock price prediction. The effectiveness of different window sizes and kernel choices is analyzed based on prediction accuracy.

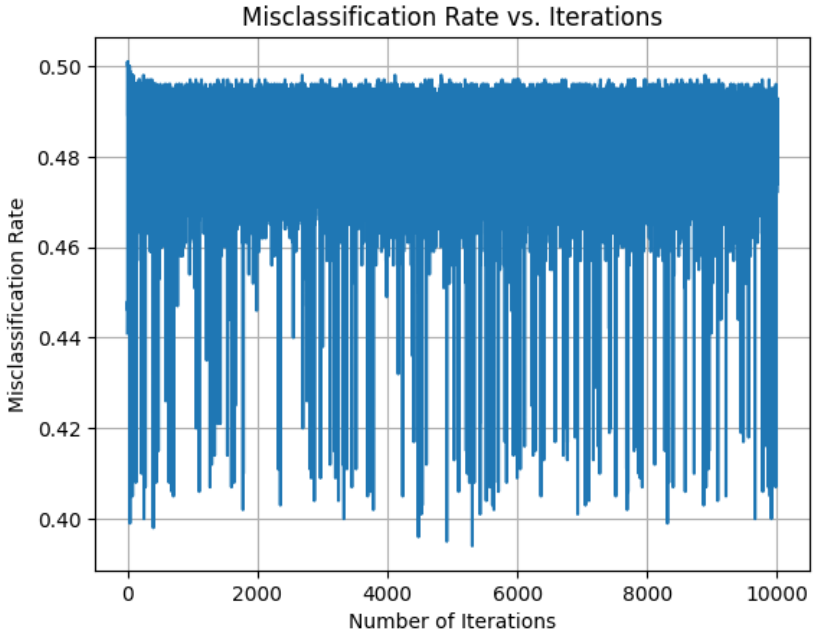

Figure 1: Misclassification rate vs. iterations for the perceptron algorithm when the max iterations are 10000

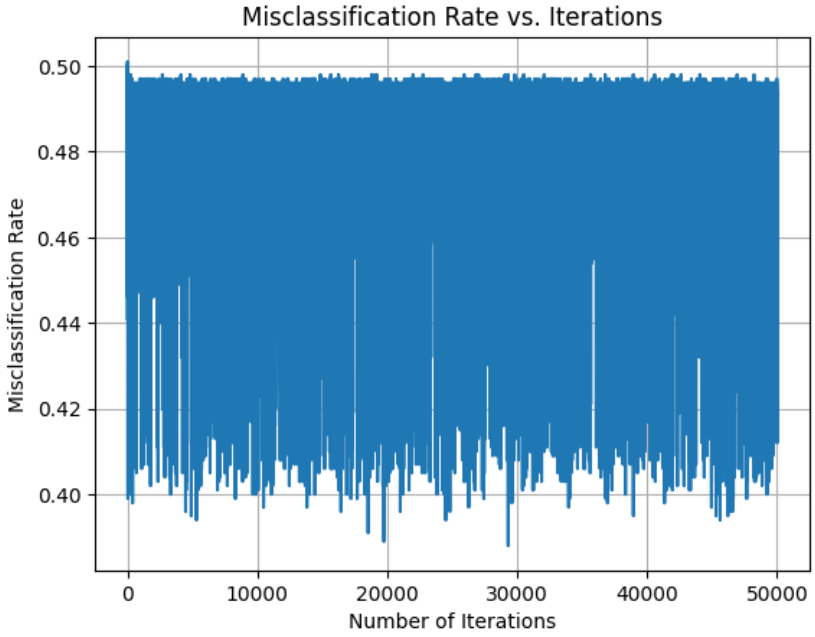

Figure 2: Misclassification rate vs. iterations for the perceptron algorithm when the max iterations are 50000

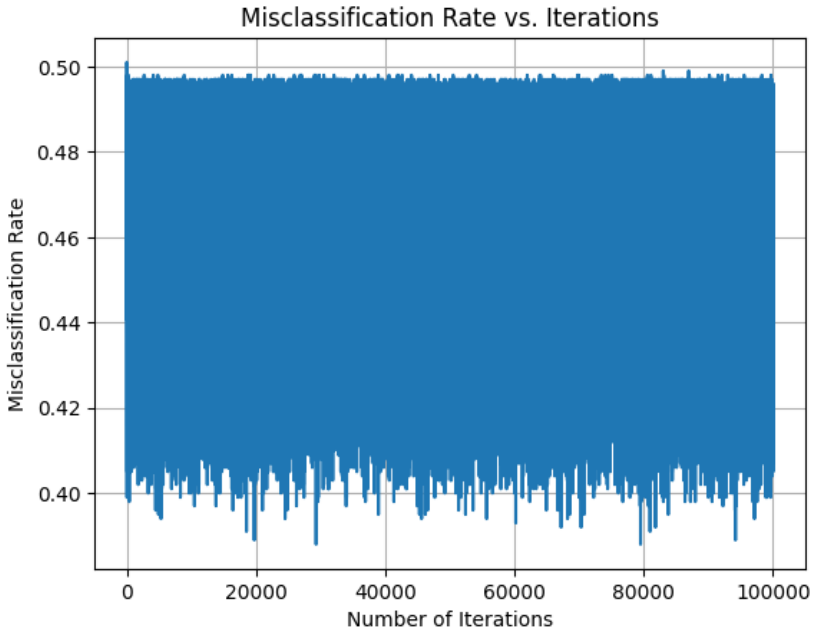

Figure 3: Misclassification rate vs. iterations for the perceptron algorithm when the max iterations are 100000

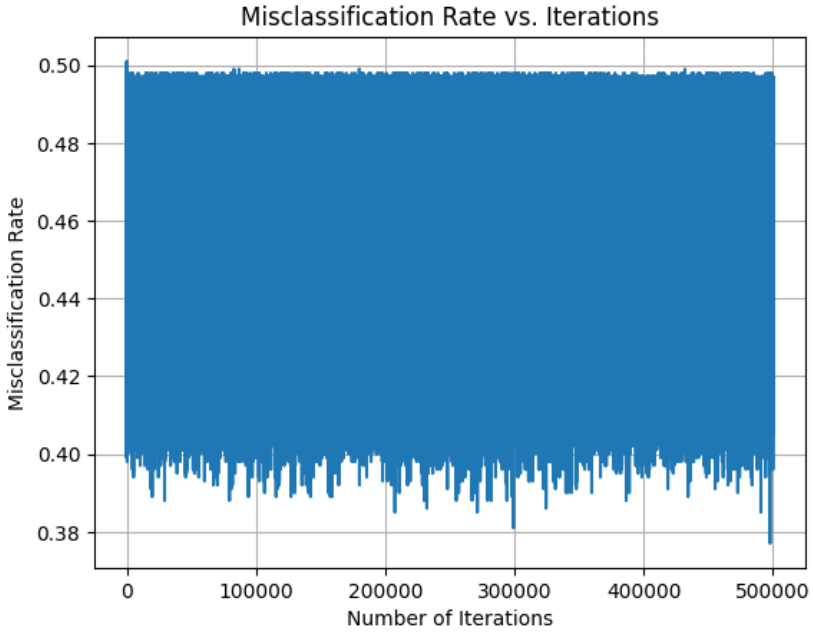

Figure 4: Misclassification rate vs. iterations for the perceptron algorithm when the max iterations are 500000

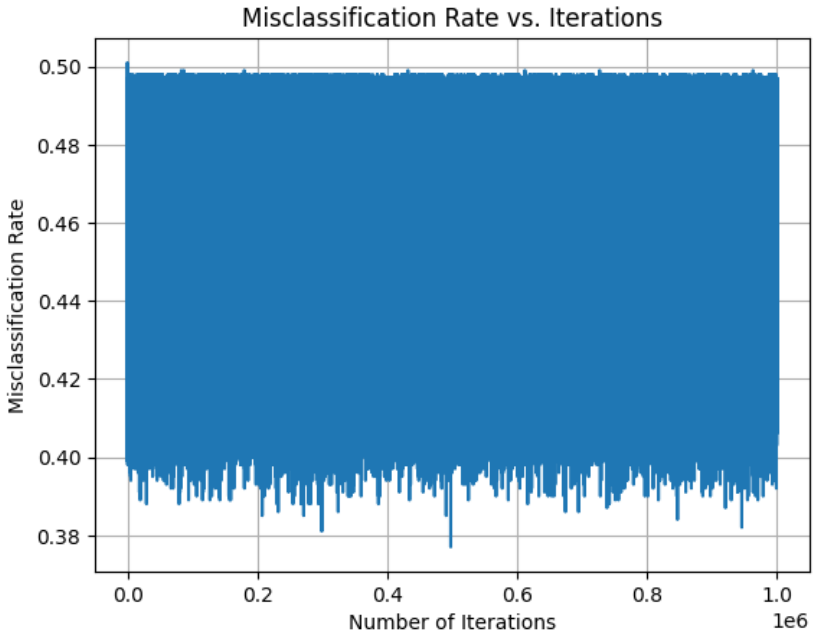

Figure 5: Misclassification rate vs. iterations for the perceptron algorithm when the max iterations are 1000000

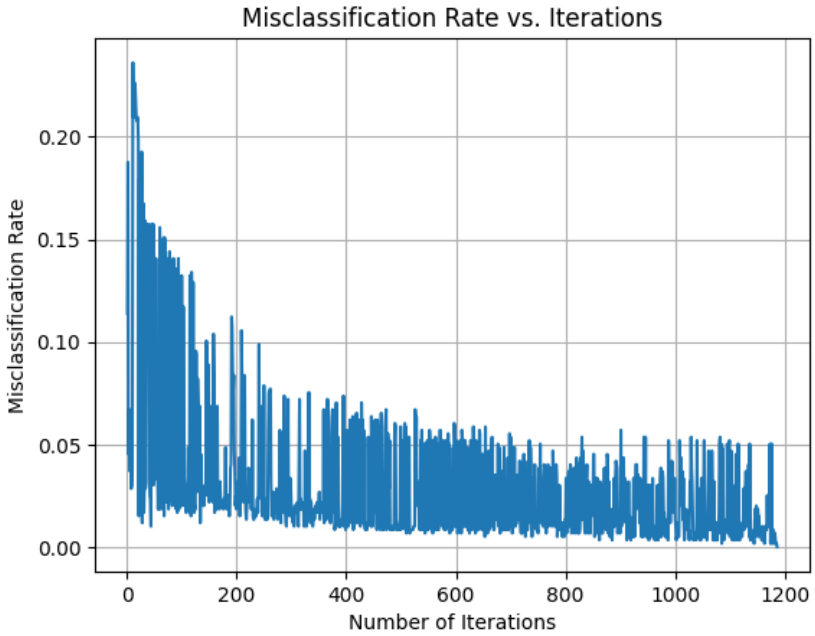

Figure 6: Misclassification rate vs. iterations after removing non-separable images.

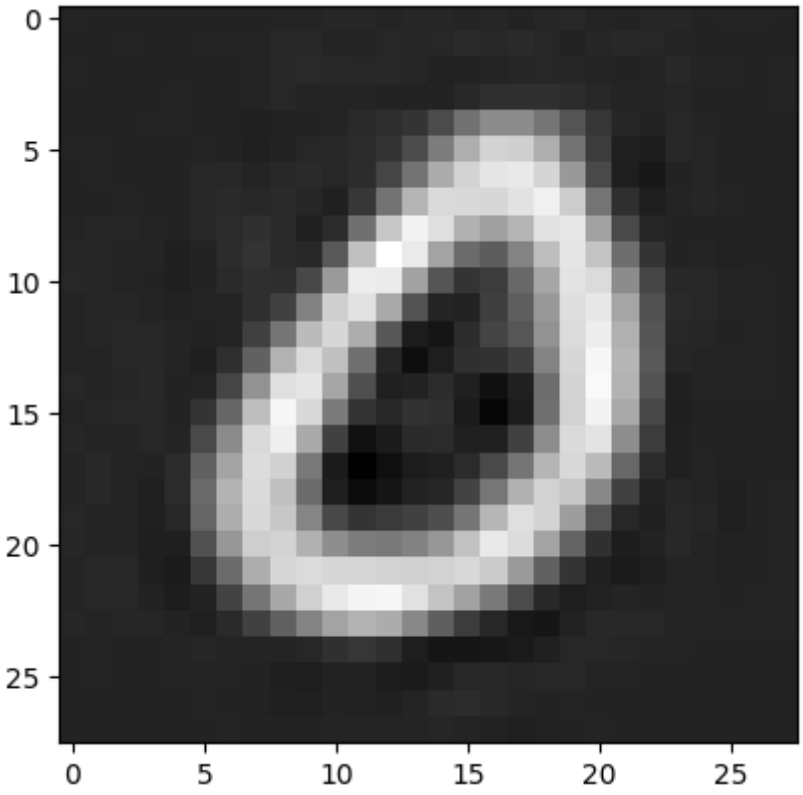

Figure 7: Reconstructed image of 0.

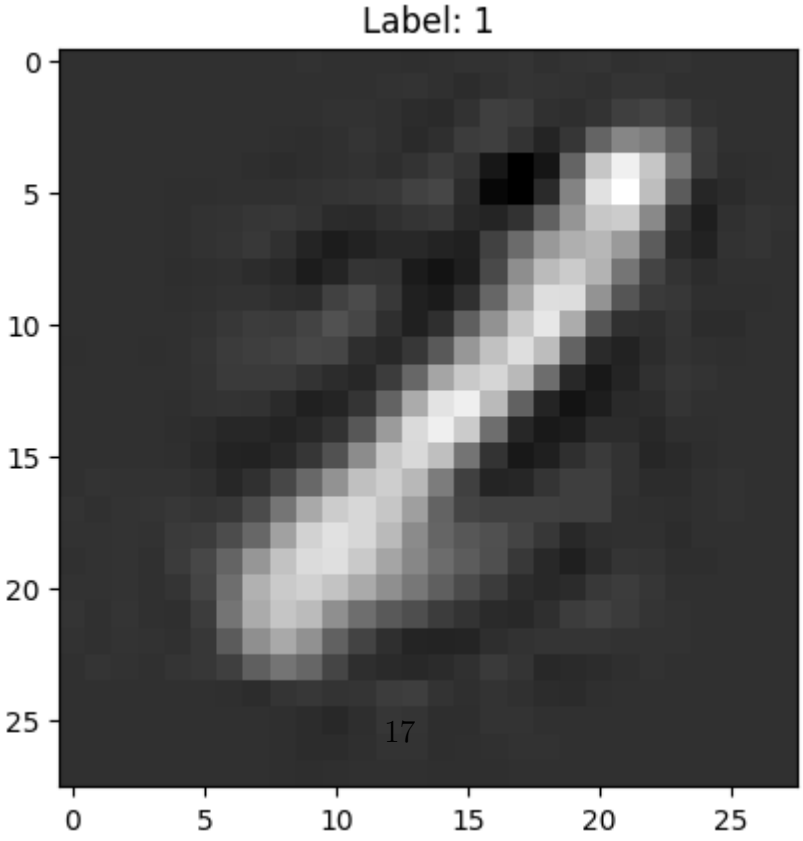

Figure 8: Reconstructed image of 1.

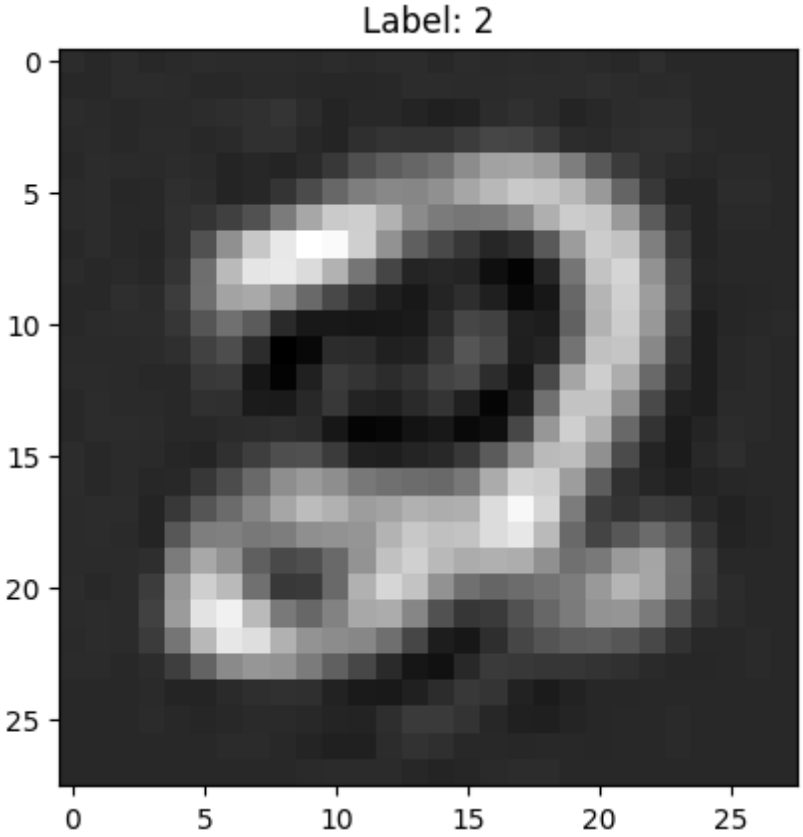

Figure 9: Reconstructed image of 2.

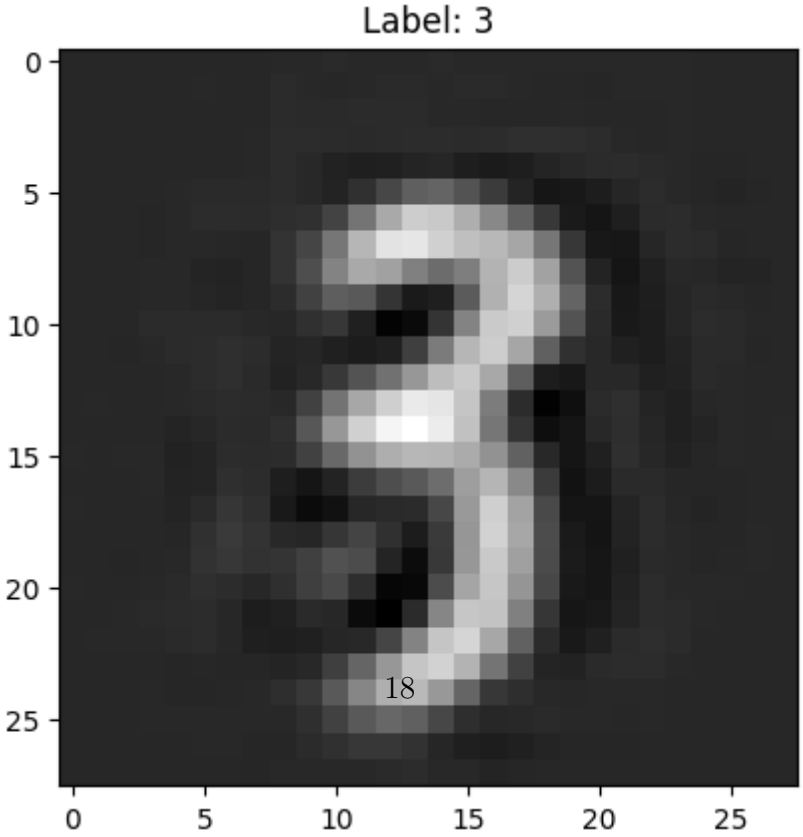

Figure 10: Reconstructed image of 3.

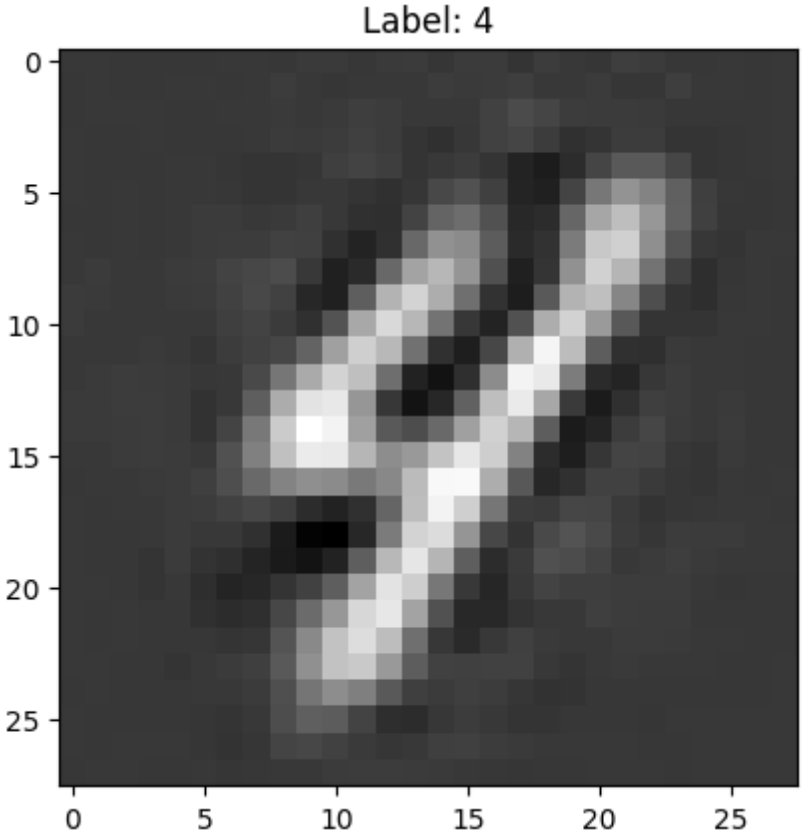

Figure 11: Reconstructed image of 4.

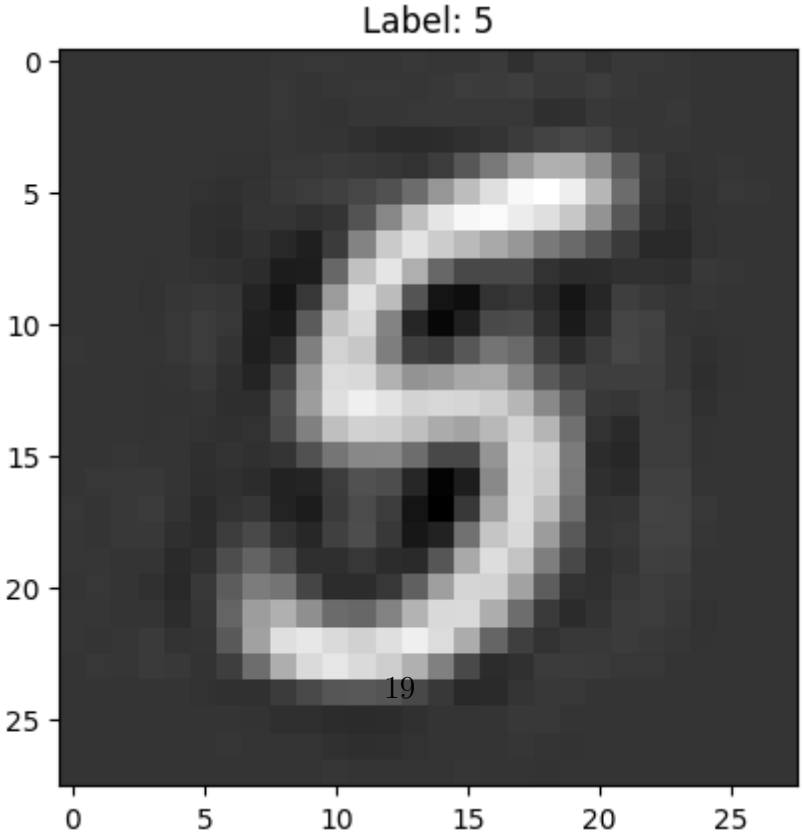

Figure 12: Reconstructed image of 5.

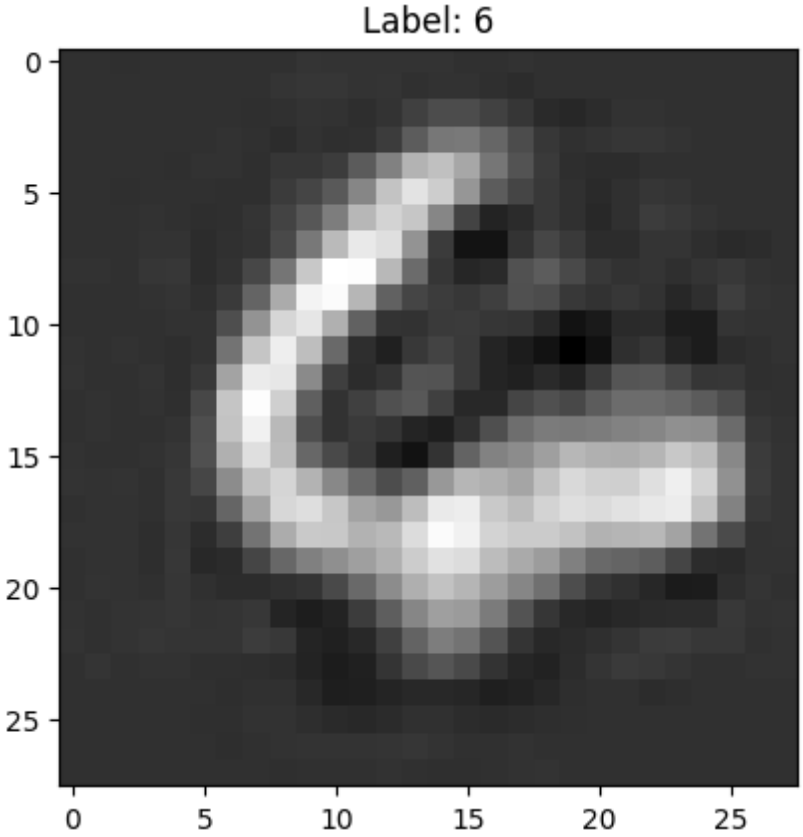

Figure 13: Reconstructed image of 6.

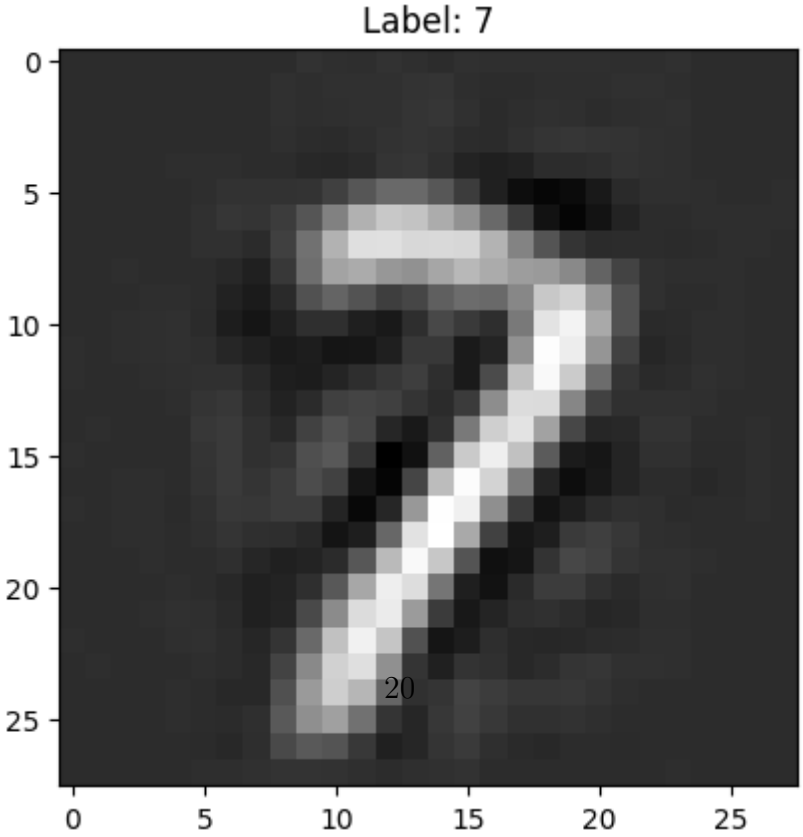

Figure 14: Reconstructed image of 7.

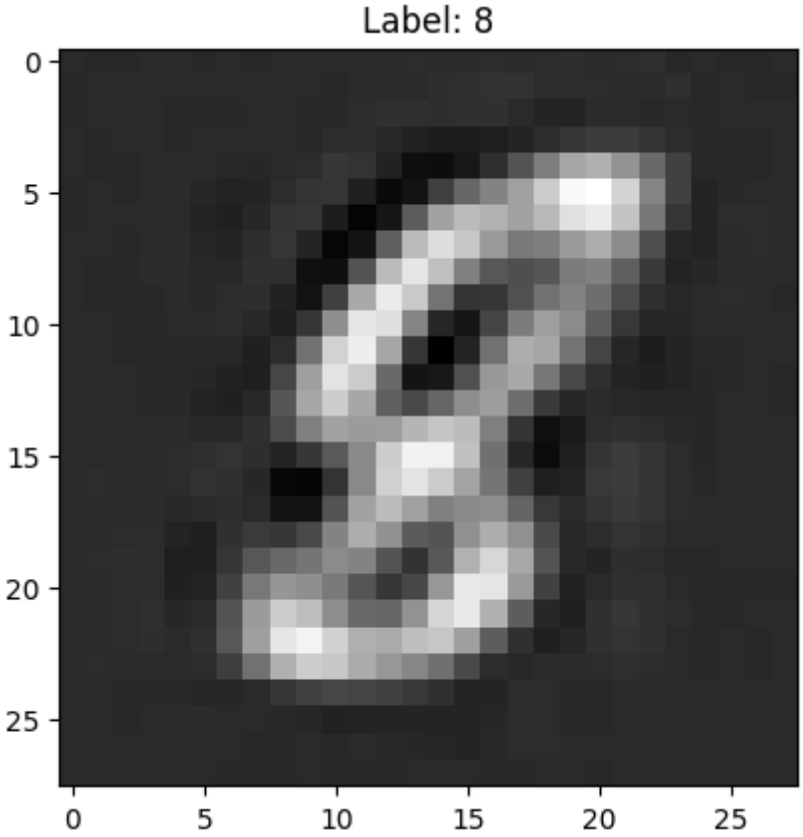

Figure 15: Reconstructed image of 8.

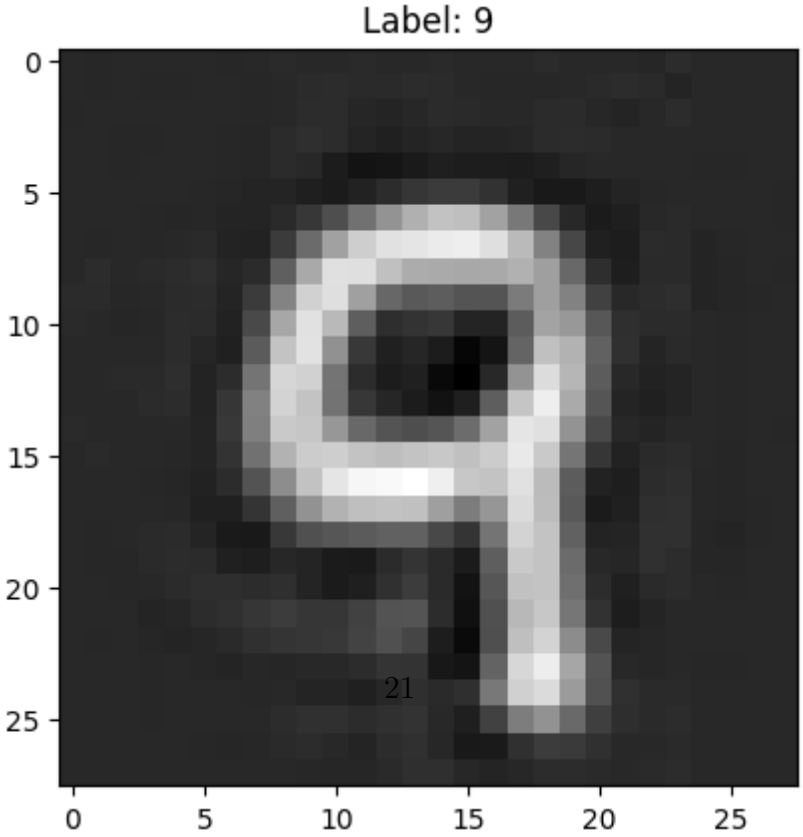

Figure 16: Reconstructed image of 9.

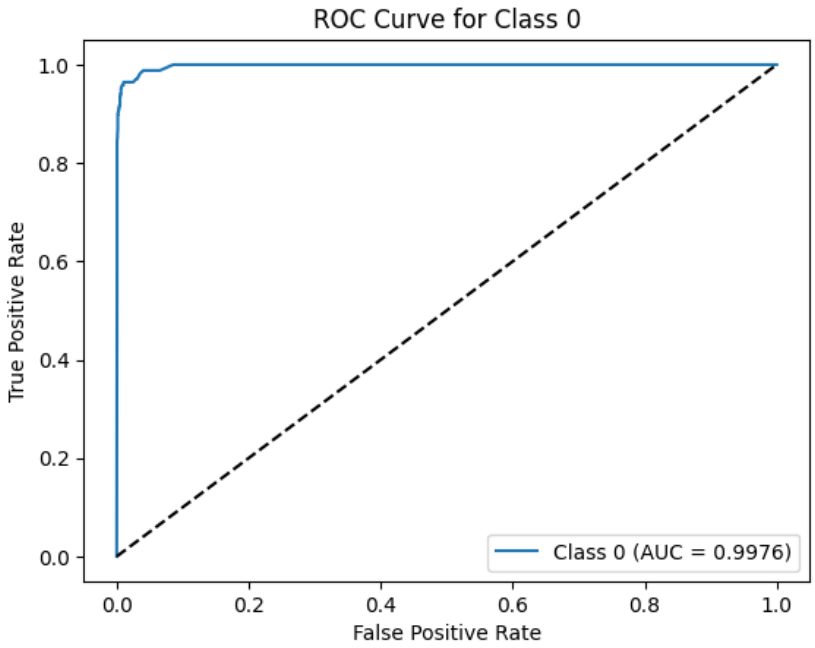

Figure 17: ROC Curve 0 vs Rest

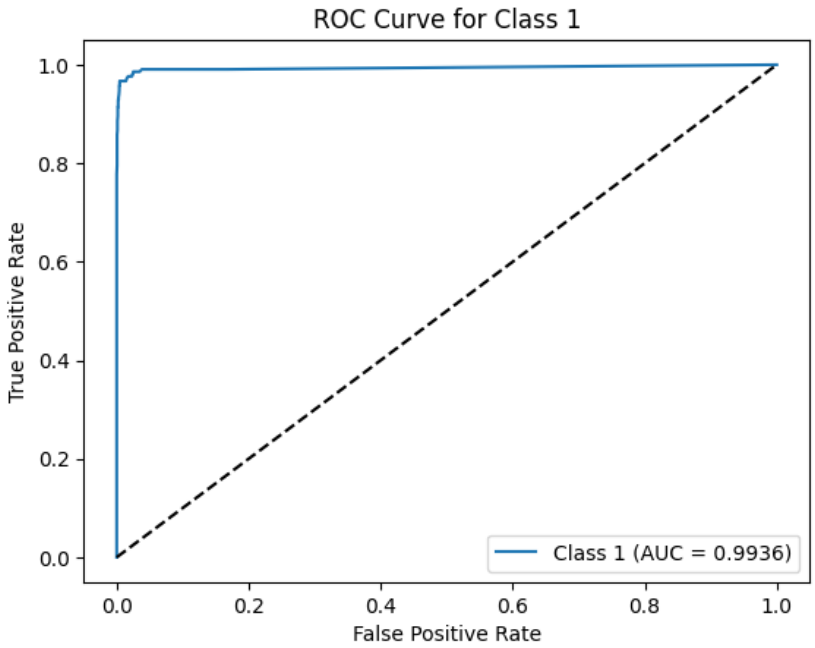

Figure 18: ROC Curve 1 vs Rest

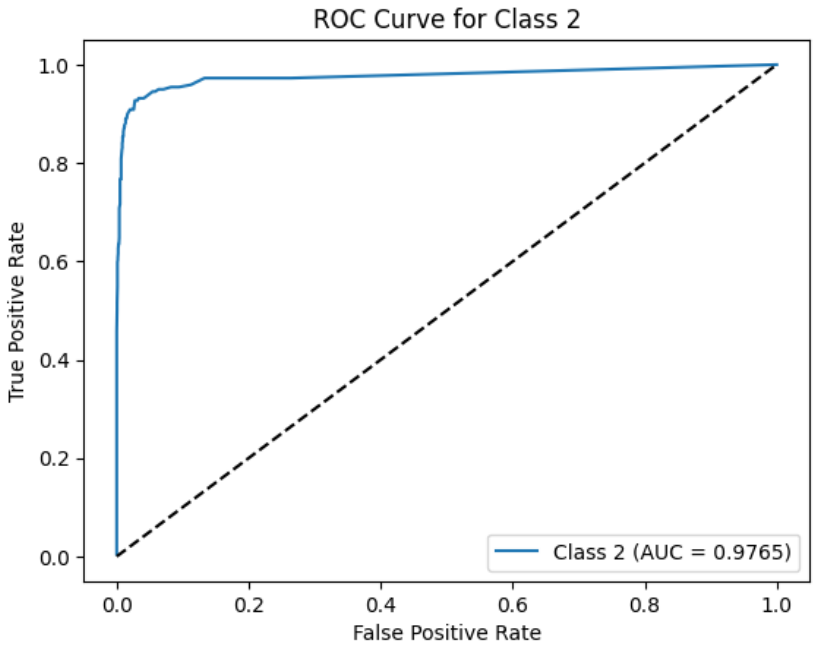

Figure 19: ROC Curve 2 vs Rest

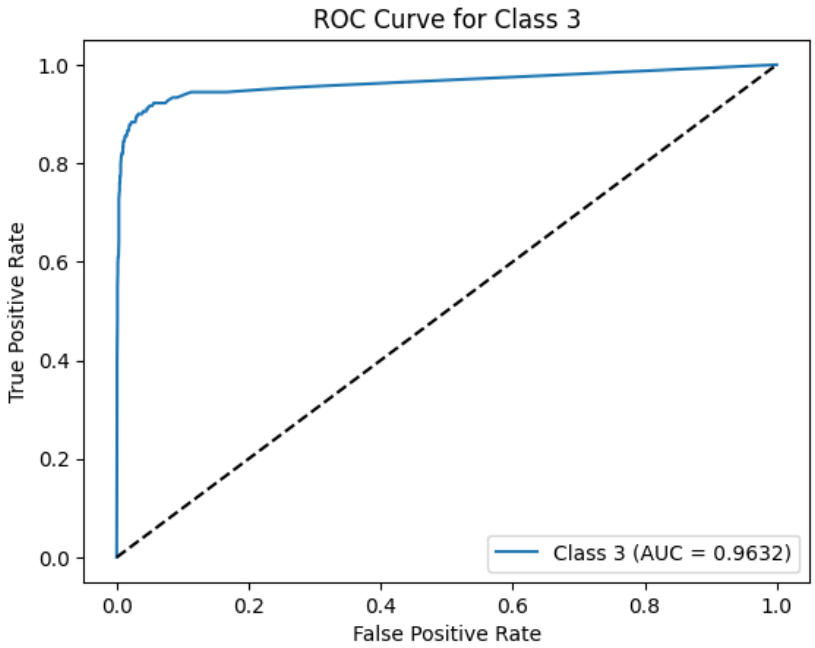

Figure 20: ROC Curve 3 vs Rest

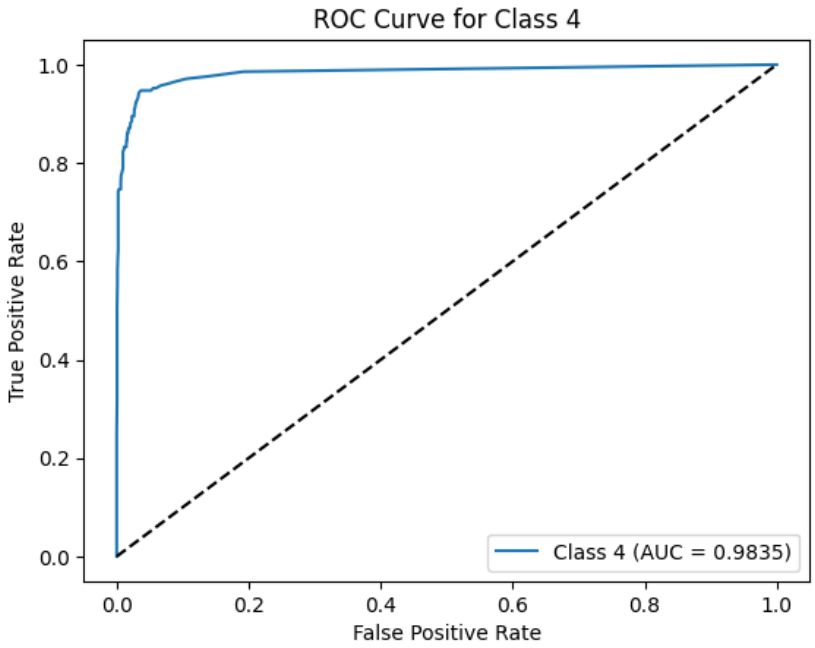

Figure 21: ROC Curve 4 vs Rest

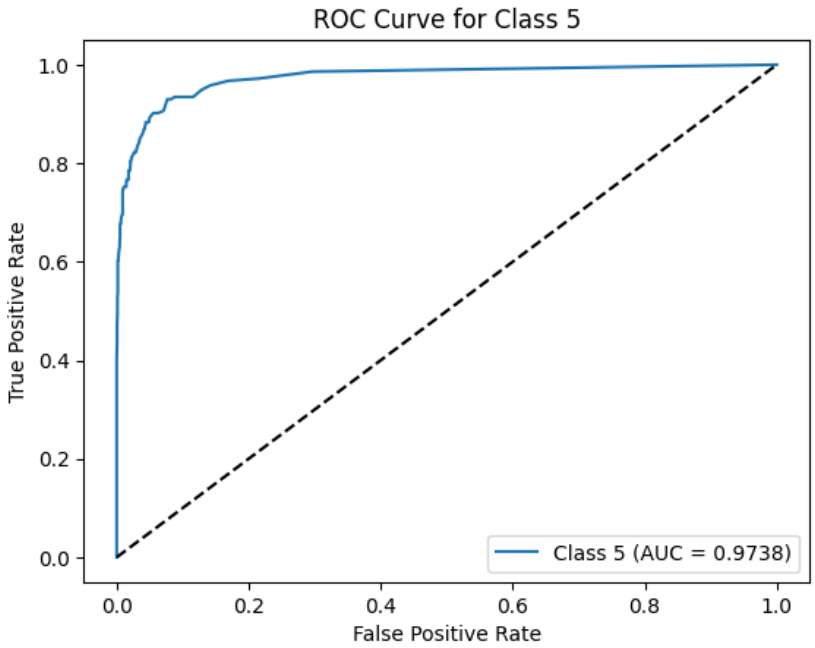

Figure 22: ROC Curve 5 vs Rest

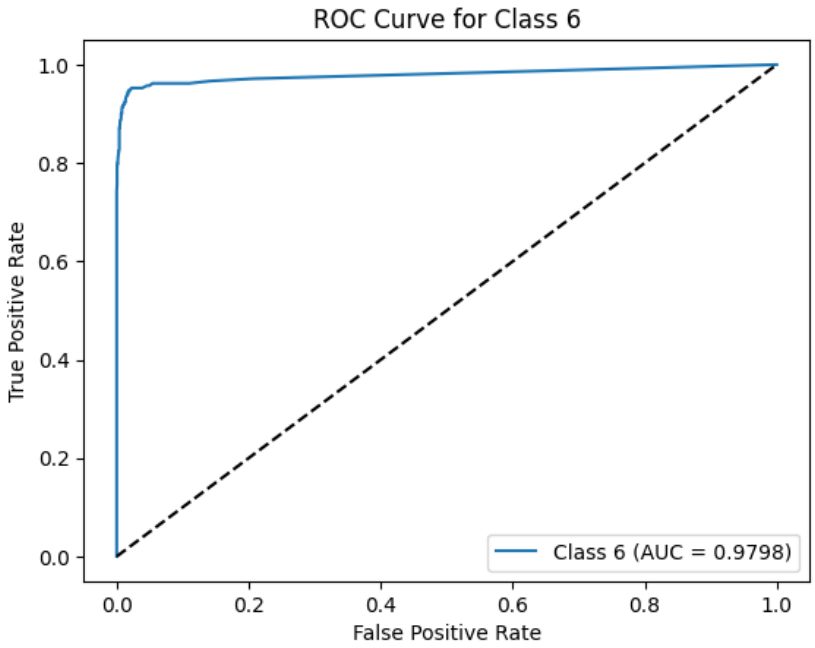

Figure 23: ROC Curve 6 vs Rest

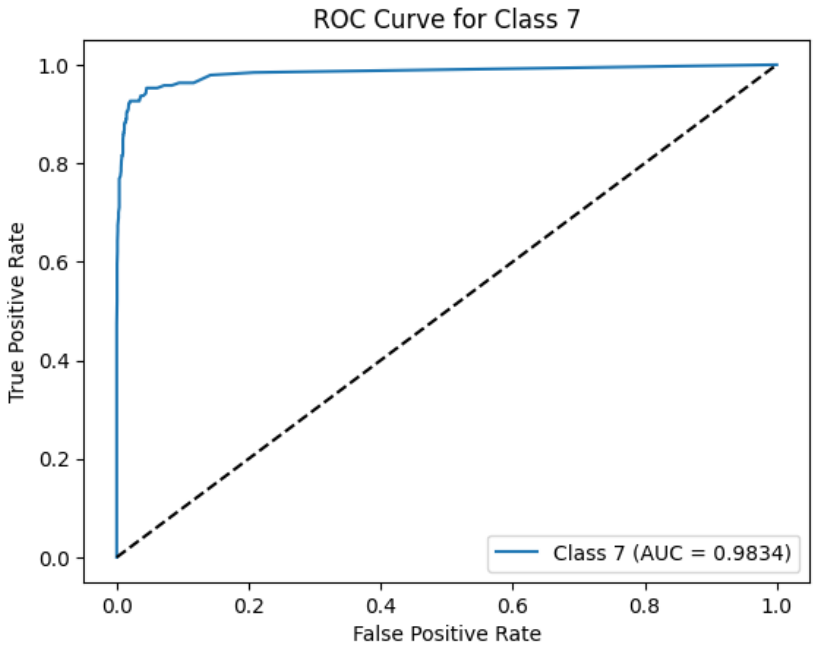

Figure 24: ROC Curve 7 vs Rest

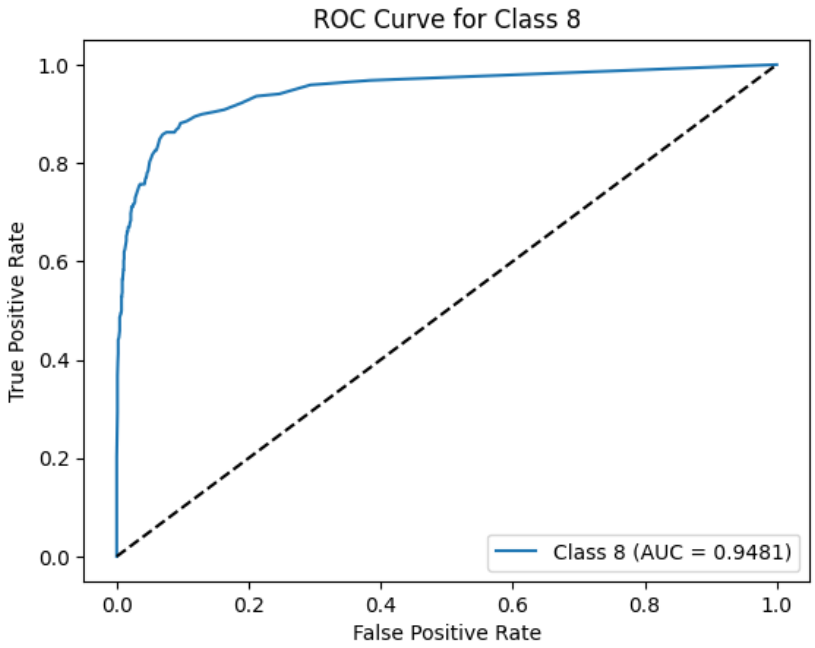

Figure 25: ROC Curve 8 vs Rest

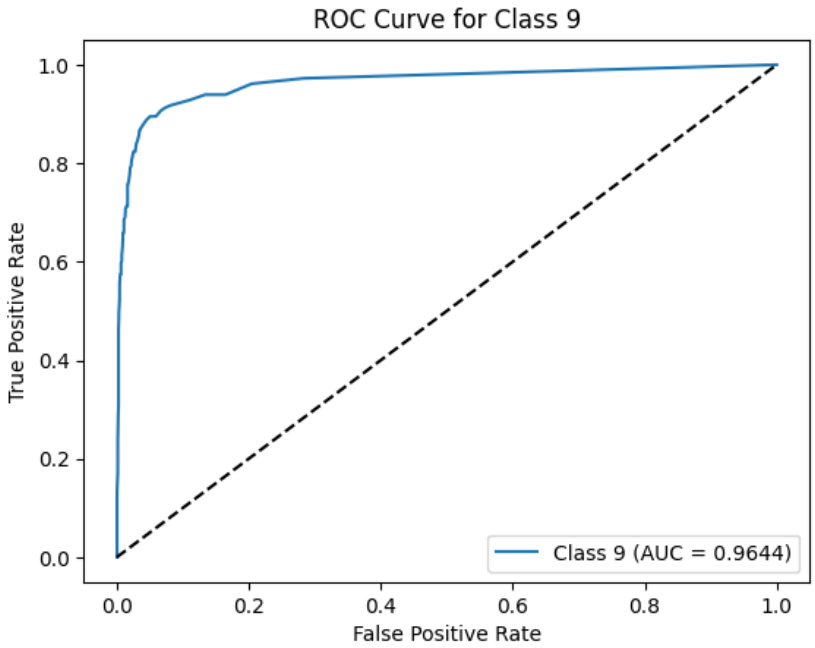

Figure 26: ROC Curve 9 vs Rest

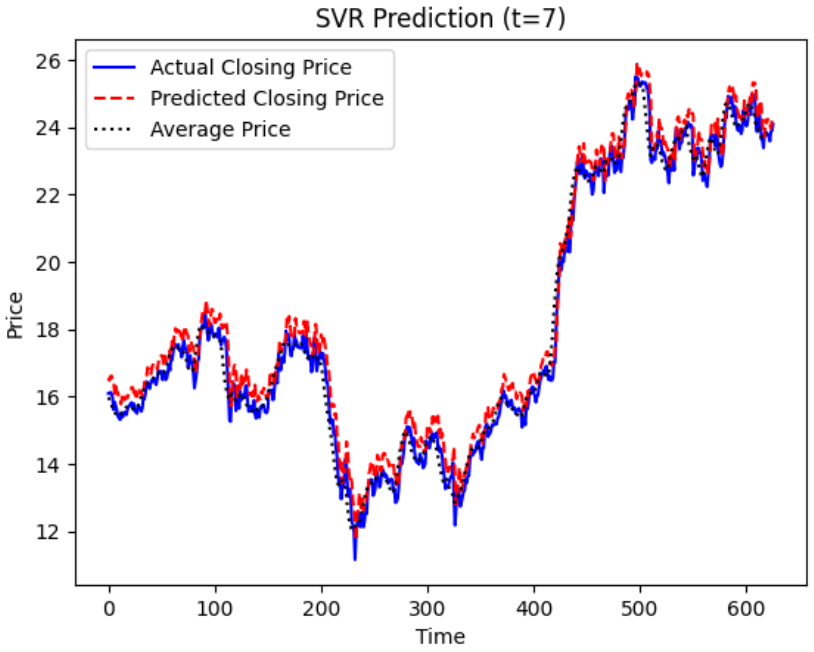

Figure 27: Linear SVR Predictor for t = 7

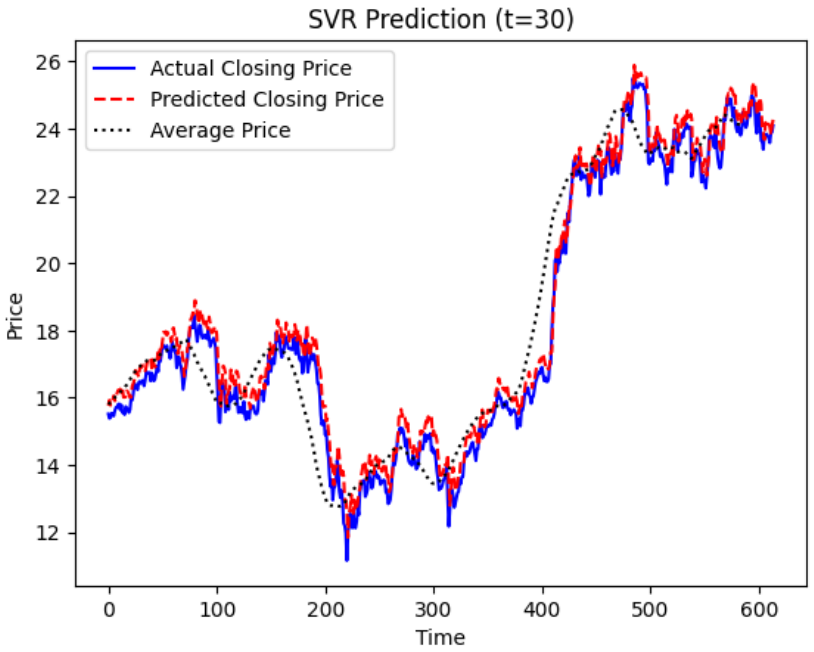

Figure 28: Linear SVR Predictor for t = 30

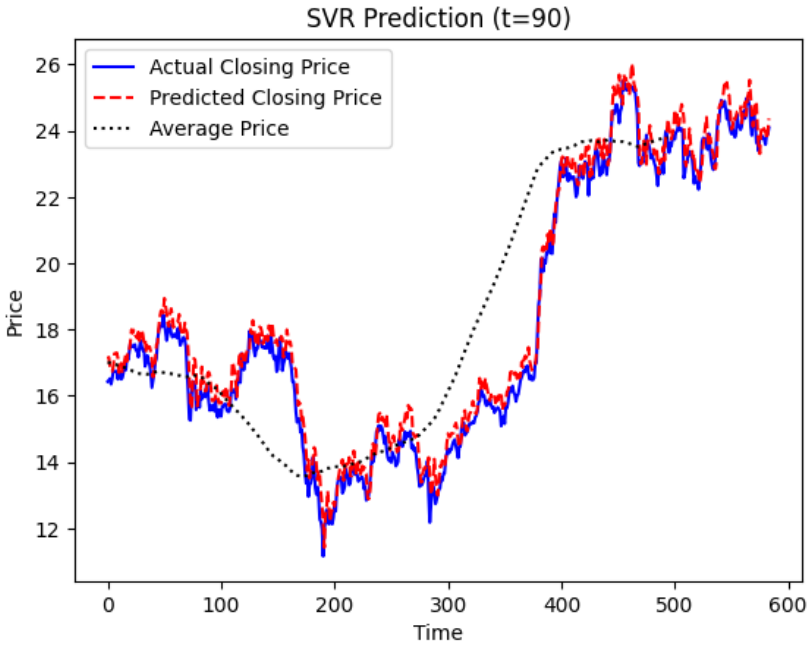

Figure 29: Linear SVR Predictor for t = 90

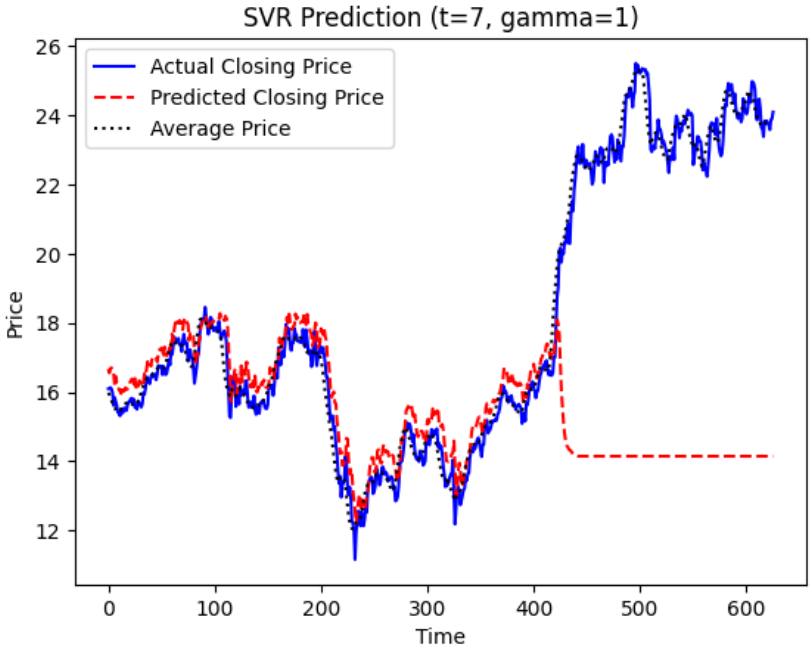

Figure 30: Gaussian SVR Predictor for t = 7 with $\gamma = 1$

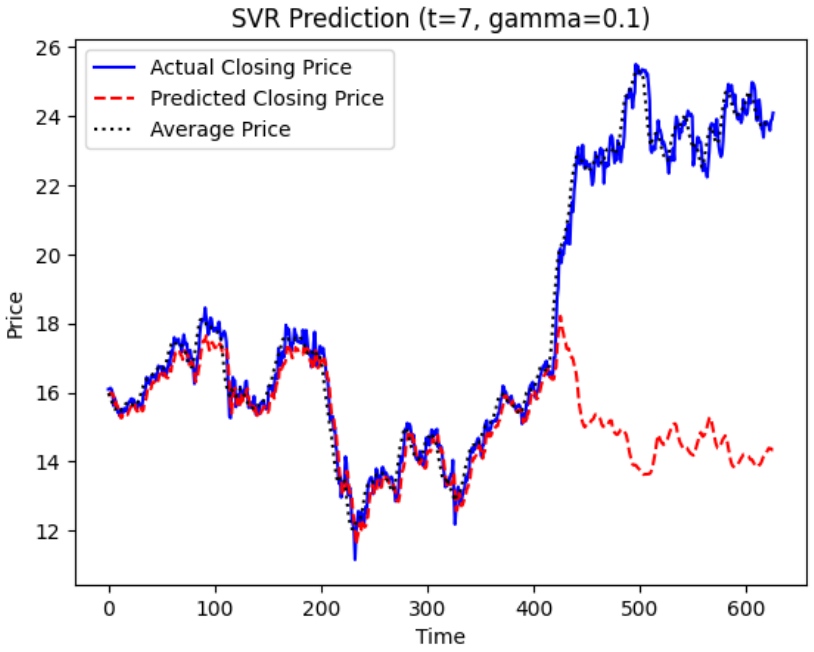

Figure 31: Gaussian SVR Predictor for t = 7 with $\gamma = 0.1$

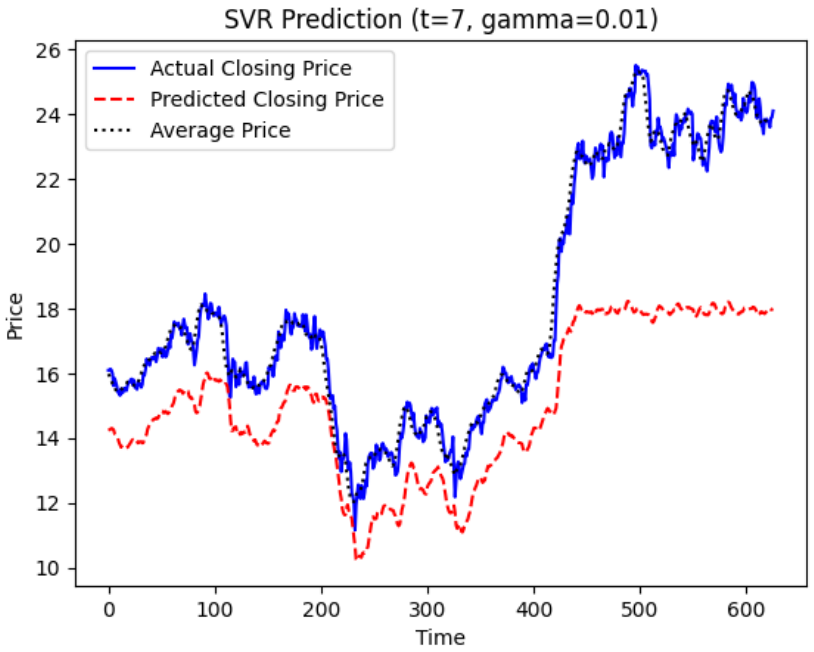

Figure 32: Gaussian SVR Predictor for t = 7 with $\gamma = 0.01$

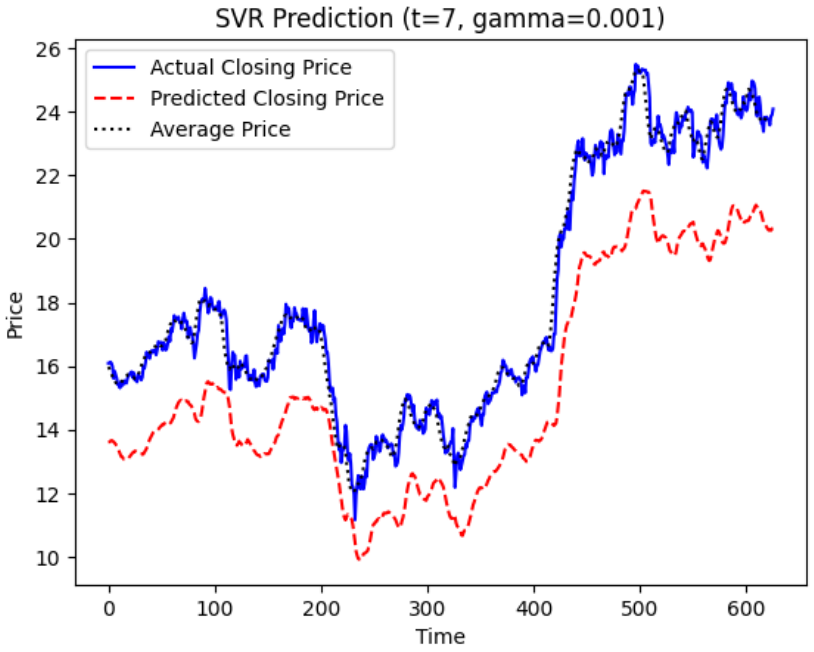

Figure 33: Gaussian SVR Predictor for t = 7 with $\gamma = 0.001$

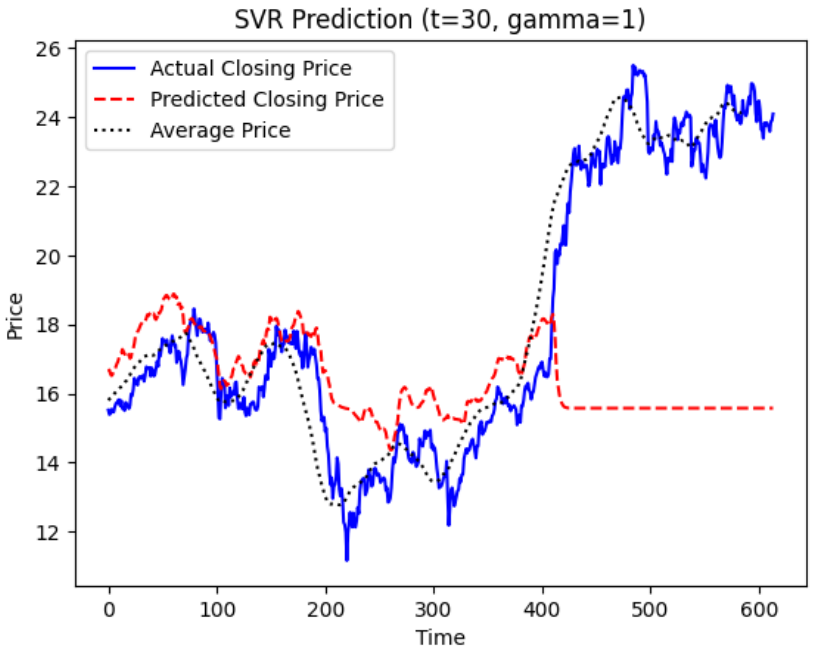

Figure 34: Gaussian SVR Predictor for t = 30 with $\gamma = 1$

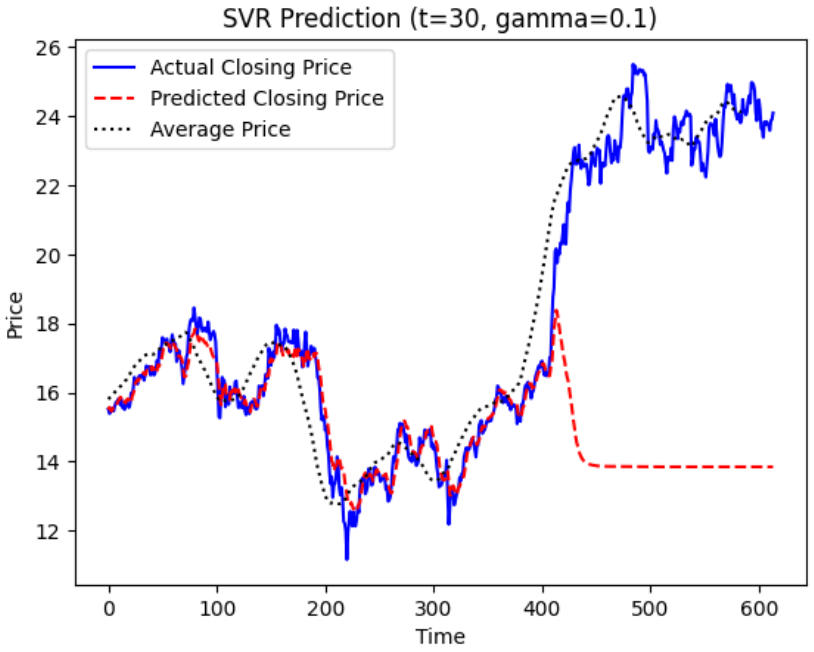

Figure 35: Gaussian SVR Predictor for t = 30 with $\gamma = 0.1$

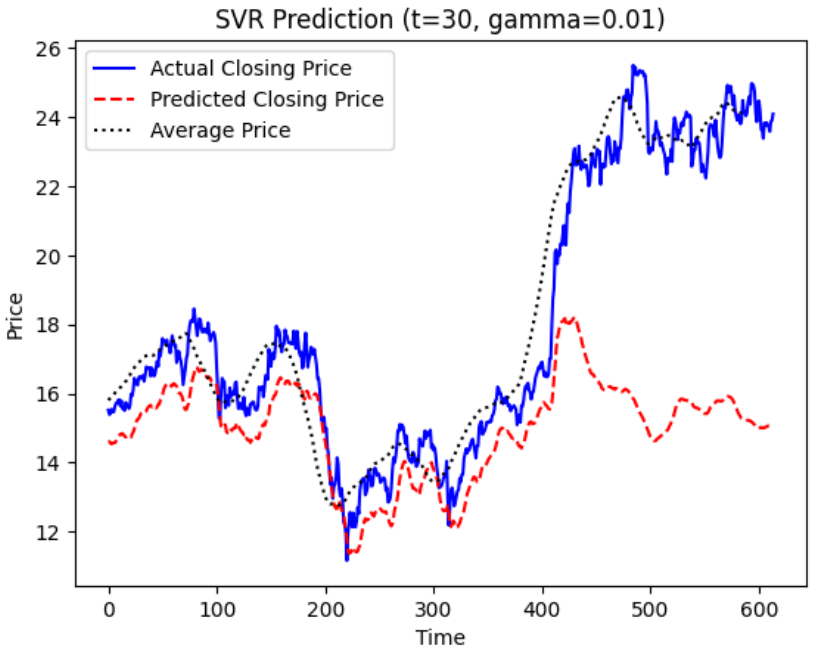

Figure 36: Gaussian SVR Predictor for t = 30 with $\gamma = 0.01$

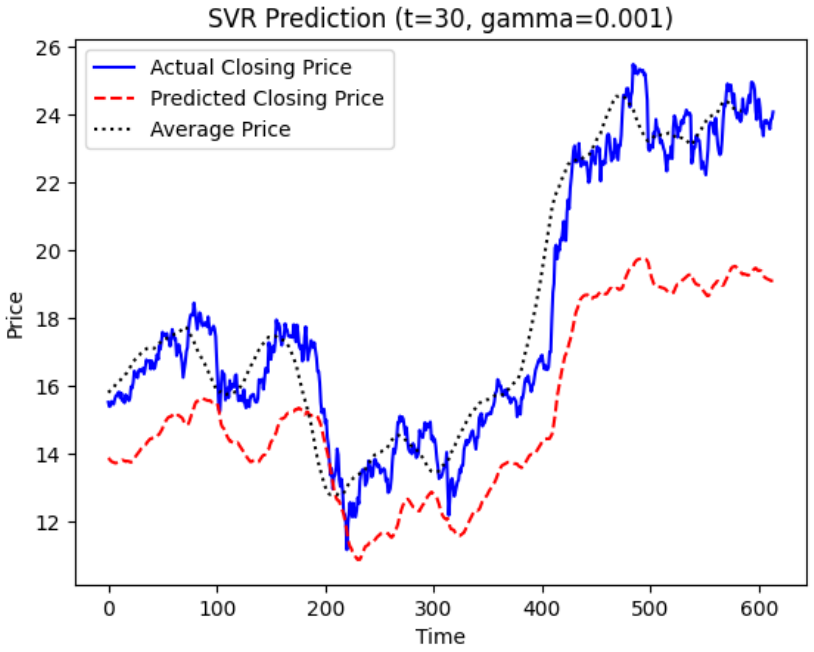

Figure 37: Gaussian SVR Predictor for t = 30 with $\gamma = 0.001$

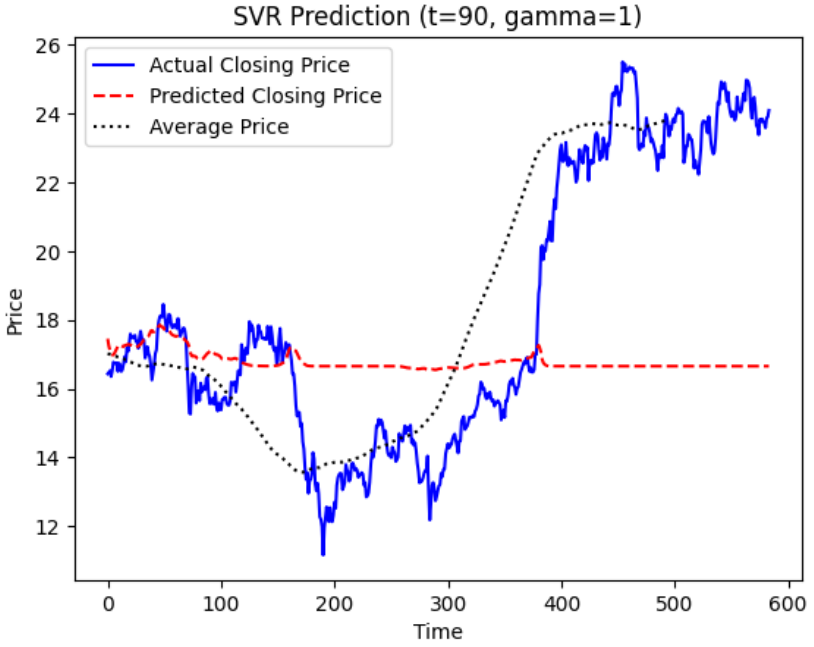

Figure 38: Gaussian SVR Predictor for t = 90 with $\gamma = 1$

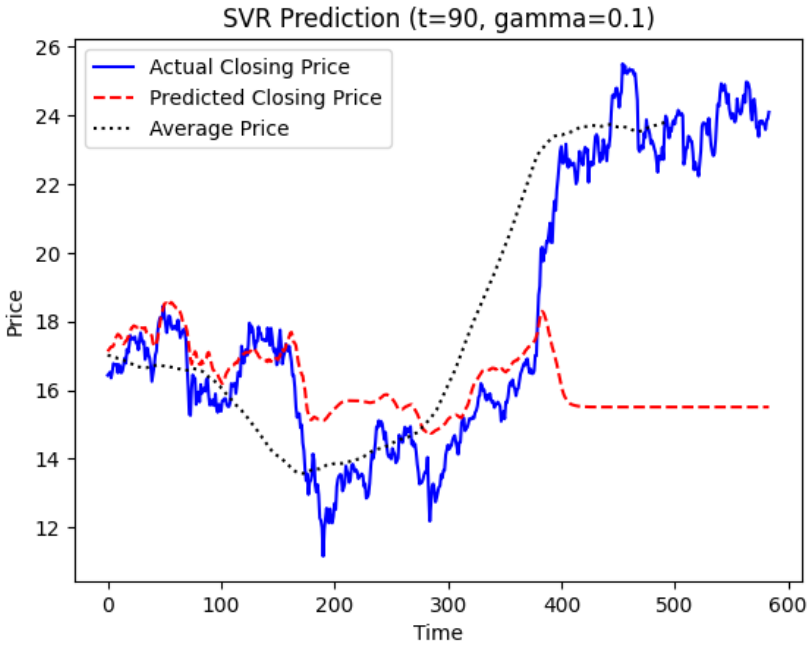

Figure 39: Gaussian SVR Predictor for t = 90 with $\gamma = 0.1$

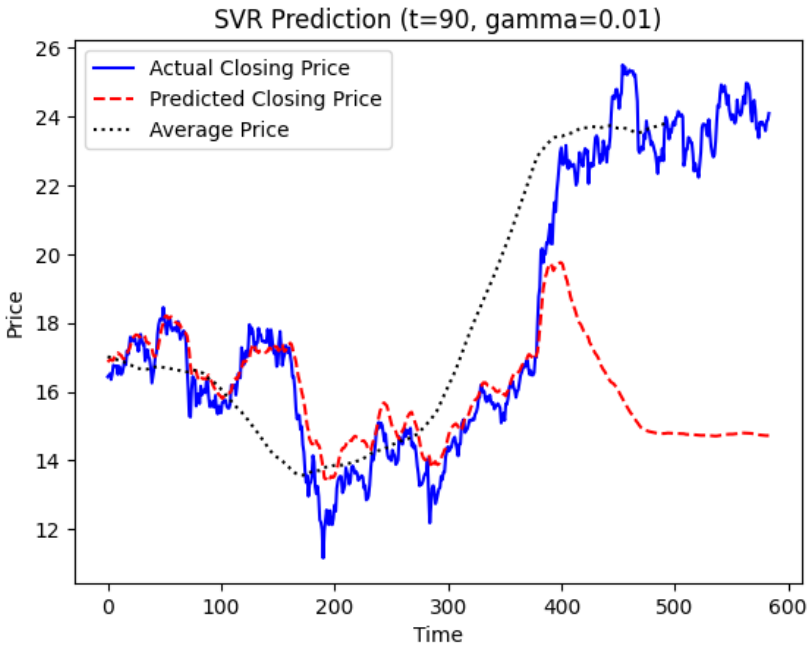

Figure 40: Gaussian SVR Predictor for t = 90 with $\gamma = 0.01$

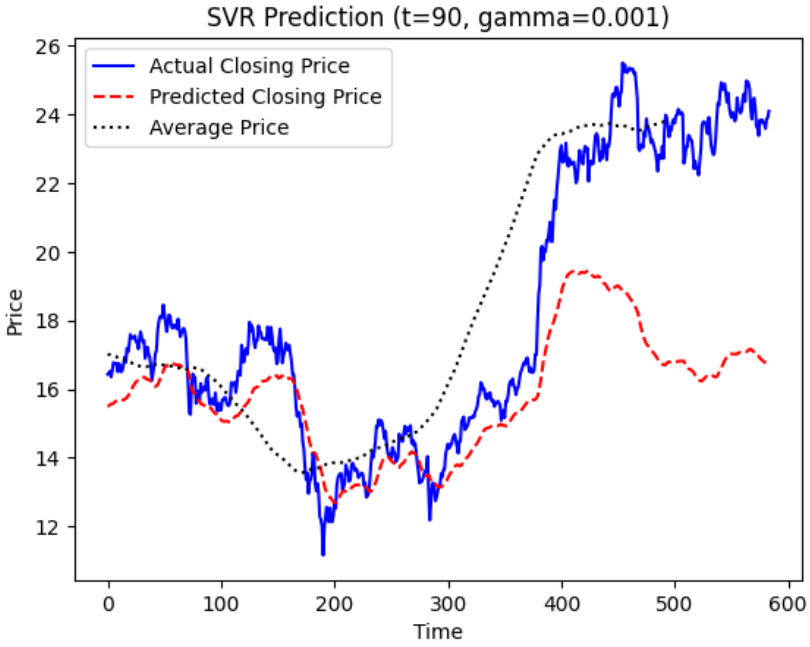

Figure 41: Gaussian SVR Predictor for t = 90 with $\gamma = 0.001$