UMC-203 Assignment 2

Sehaj Ganjoo
sehajganjoo@iisc.ac.in
Sr No: 23651

March 28, 2025

1 Question 1: Support Vector Machine and Perceptron

The dataset used here is the CIFAR-100, which contains 100 classes of images. I first querried the Oracle and cached the data using file handling.

Overview of the dataset recieved:

```
Train Data Size: 1000, Test Data Size: 200
Type of train_data[0][0]: <class 'list'>
Type of train_data[0][1]: <class 'float'>
```

So I first converted the data into numpy arrays for easier manipulation.

```
Train Features Shape: (1000, 27)
Train Labels Shape: (1000,)
Test Features Shape: (200, 27)
Test Labels Shape: (200,)
```

Task 1: Perceptron

I implememted the Perceptron algorithm and run it on the dataset, with maximum iteration set to "100000".
The output was:

```
The Perceptron Algorithm has not converged after 10000 iterations
```

We can make certain that the algorithm doesnt converge by looking at the plot of the Error count for each iteration. The error count is basically, the number of misclassified points in the dataset for each update.

Deliverable 1:

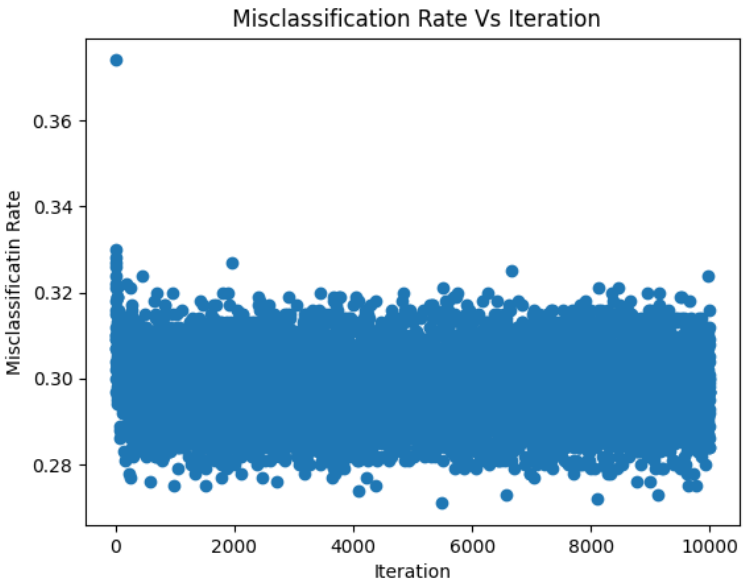


Figure 1: Misclassifition Rate vs Iteration

The error rate **did not converge to zero**, even after increasing the number of iterations. The error plot remained more or less the same, indicating that the Perceptron was unable to find a separating hyperplane.
Hence, it is not able to converge on the dataset.
This strongly suggests that the dataset is not linearly separable. Because if it were, the Perceptron will converge in a finite number of updates and achieve zero classification error.

Task 2: Linear Support Vector Machine

We are required to implement the Linear SVM in both primal and dual form, by using cvopt and numpy.

Primal Form:

The primal form of the SVM is given by the following optimization problem:

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

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

$$\xi_i \geq 0, \quad \forall i. \tag{3}$$

I took C=10.

Dual Form:

The dual form of the SVM is given by the following optimization problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{4}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C, \quad \forall i, \tag{5}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{6}$$

We transform them into a quadratic programming problem and solve it using cvxopt.

$$\min_{x} \quad \frac{1}{2}x^T P x + q^T x \tag{7}$$

subject to:

$$Gx \leq h, \quad Ax = b \tag{8}$$

where the decision variable is:

$$x = \begin{bmatrix} w \\ b \\ \xi \end{bmatrix}$$

For Task 2, I implement the Linear SVM in both primal and dual form.
The results are as follows:

```
For Primal
Weights:[ 2.31506567 -5.61372798 -0.22023752  1.39444312
   -5.99611468 -1.25348955
2.00222775 -0.55679986  2.37882948 -0.52557532  4.05950934
   0.83478054
-0.21472826  0.2015062   1.79034733  1.25945118 -1.68636491
   0.44786271
-0.0973282  -1.16116704  0.99720019 -0.12642016  1.91866561
   -0.28675904
-1.18359086 -0.48654774 -0.59031803]
Bias: 0.5852925285329721

For Dual
Weights:[ 2.31506567 -5.61372798 -0.22023752  1.39444312
   -5.99611468 -1.25348955
2.00222775 -0.55679986  2.37882948 -0.52557532  4.05950934
   0.83478054
-0.21472826  0.2015062   1.79034733  1.25945118 -1.68636491
   0.44786271
-0.0973282  -1.16116704  0.99720019 -0.12642016  1.91866561
   -0.28675904
-1.18359086 -0.48654774 -0.59031803]
Bias: 0.659416539922803
```

Deliverable 2:

Let us look at the runtime of both the computations. The time taken is calculated using the time library in python.

```
>>>Time taken for Primal SVM: 1.6479098796844482
>>>Time taken for Dual SVM: 1.288203477859497
```

The time taken for both the computations is almost the same.
Primal SVM performs better than Dual SVM in terms of time taken for computation.

Primal SVM,requires solving a QP problem with a matrix of size $(D+1+N)\times(D+1+N)$, where:

- D is the number of features,
- N is the number of training samples.

Dual SVM requires solving a QP problem with a matrix of size $N \times N$.

Both have the same number of inequality constraints.

However, in the primal formulation, slack variables are explicitly included, increasing the number of optimization variables.

Hence, dual computation has a slight edge over the primal in this case, as it only depends on the support vectors.

Let us confirm our intuition by running both the SVMs on test data for 50 iterations and record the time taken.

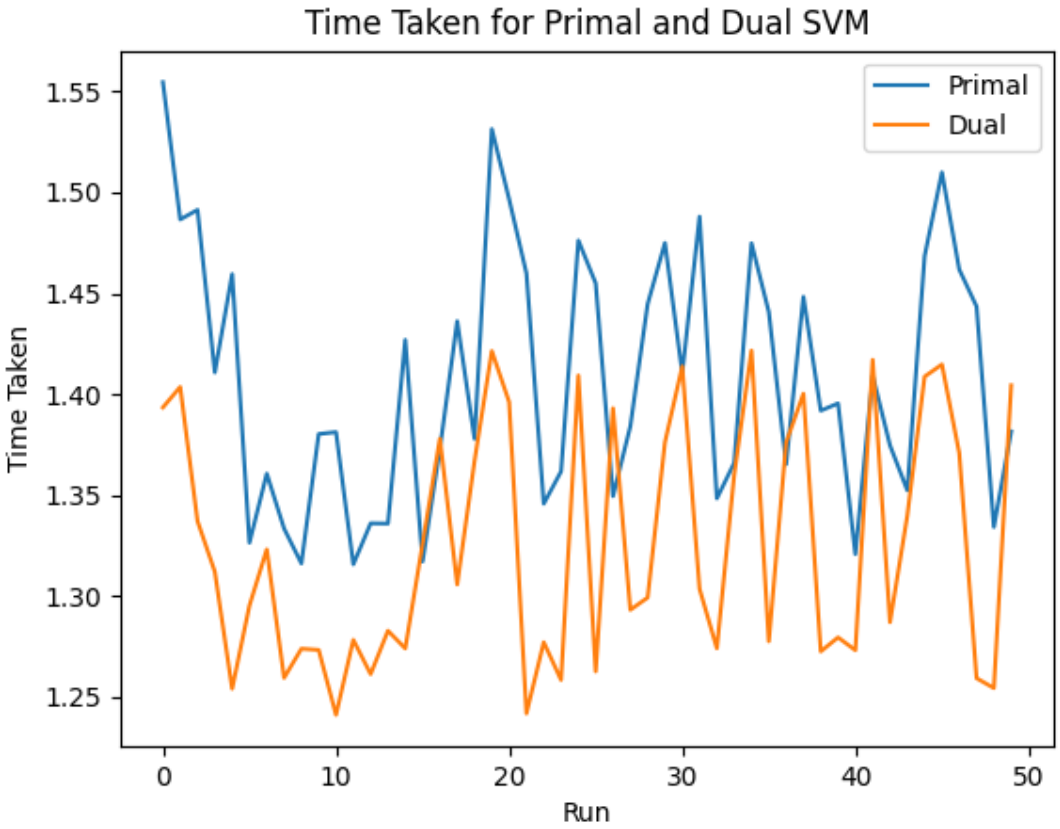


Figure 2: Time taken for Primal and Dual SVM

We see that dual SVM performed better than primal SVM consistently, however the difference is not highly significant Even the gap between time taken for each iteration has no specific trend. We observe a bit of randonmness and that maybe due to resource alloaction and quad prog efficiency.

Deliverable 3:

For Task 2, we also checked and seperated those point which are causing the data to be non-separable. So,basically we have to those points which are "Outliers" in the dataset and isolate them. Outliers are of two types:

1. Support Vectors within the Margin($0 < \xi_i \leq 1$): These points lie inside the margin but are still correctly classified. But are nonetheless considered an outlier, as for a point to not be an outlier, it should be positioned on the correct side of the appropriate marginal hyperplane.

2. Misclassified Points (Severe Outliers)($\xi_i > 1$): These points lie on the wrong side of the decision boundary and are misclassified.

If we omit the outliers, the training data is correctly separated. So we indeed get a linearly seperable data. But, we really dont need to remove the support vectors, as they don't actually contribute to the classifier even in the linearly seperable case and lie exactly on the margin.
Therefore to obtain a linearly seperable data, we need to remove the misclassified points.

```
primal case
Number of Non-Seperable Points: 496
Number of Misclassified Points: 177
misclassified inidices: [  1   2   4   6   8   9  11  17  23  36
   39  40  50  51  55  70  72  75 86  93  99 117 118 124 130 138
   141 150 152 158 169 171 184 185 202 206 211 212 214 224 230 238
   244 248 252 260 268 277 279 283 285 292 295 302 311 312 318 322
   325 326 335 336 343 347 351 353 356 383 389 391 415 427 429 431
   438 449 452 453 455 460 461 493 494 498 500 507 513 515 516 520
   521 525 529 532 543 545 546 556 557 574 580 584 589 594 596 603
   606 610 611 615 626 631 632 639 641 643 648 651 657 665 670 677
   684 688 691 692 720 723 725 726 747 755 766 769 781 783 784 785
   787 790 794 800 809 816 822 829 832 835 839 841 845 848 866 870
   875 877 878 881 885 890 897 907 909 912 916 919 927 930 931 939
   948 958 960 961 980 993 996]

dual case
Number of Non-Seperable Points: 501
Number of Misclassified Points: 178
misclassified inidices: [  1   2   4   6   8   9  11  17  23  36
   39  40  50  51  55  70  72  75 86  93  99 117 118 124 130 138
   141 150 152 158 169 171 184 185 202 206 211 212 214 224 230 238
   244 248 252 260 268 277 279 283 285 292 295 302 311 312 318 322
   325 326 335 336 343 347 351 353 356 383 389 391 415 427 429 431
   438 449 452 453 455 460 461 493 494 498 500 507 513 515 516 520
   521 525 529 532 543 545 546 556 557 574 580 584 589 594 596 603
   606 610 611 615 626 631 632 639 641 643 648 651 657 665 670 677
   684 688 691 692 720 723 725 726 747 755 766 769 781 783 784 785
   787 790 794 800 809 816 822 829 832 835 839 841 845 848 866 870
   875 877 878 881 885 890 897 907 909 912 916 919 927 930 931 939
   948 958 960 961 980 993 996]
```

Task 3: Kernelized SVM

We repeat the previous construction, but now with the Gaussian Kernel.
The Gaussian Kernel is given by:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right) \tag{9}$$

We are supposed to choose the hyperparameters C and γ in such a way that non-seperability is no longer an issue.

So, I tried a bunch of values, and found out that $C = 12$ and $\gamma = 10$ are the best values, as they gave zero training set error.

```
C: 12
Gamma: 10
Number of support vectors: 951
Accuracy of Gaussian Kernel SVM on Training Data: 100.0
Misclassification Rate of Gaussian Kernel SVM on Training Data: 0.0
```

Then we run our SVM on the test dataset.

Deliverable 4:

The results are as follows:

```
Accuracy of Gaussian Kernel SVM: 79.5
Misclassification Rate of Gaussian Kernel SVM: 20.5
```

Task 4: Perceptron on seperable data

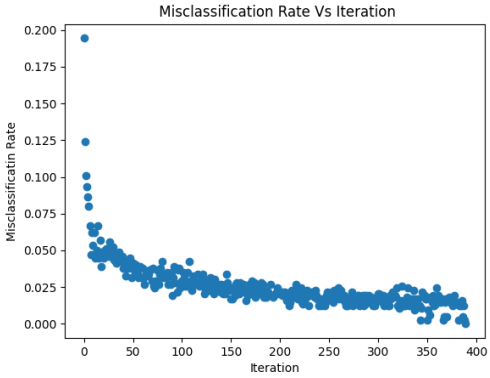
For this task, we are required to first remove the non- seperable points from out dataset and then run the perceptron on it.
Since the dataset now obtained is linearly seperable, the perceptron will converge in a finite number of iterations.

Let us do this for both the datasets obtained by removing non-seperability reported by Primal and Dual Linear SVMs.

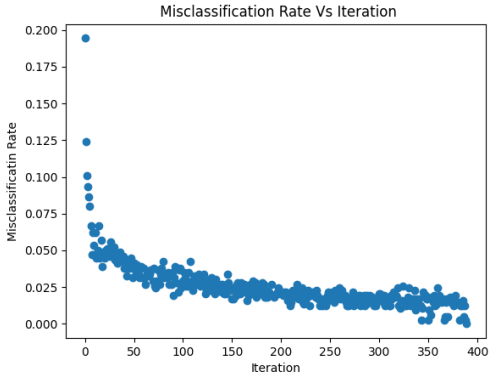
Deliverable 5:

```
Linear SVM Primal
On removing all outliers:
The Perceptron Algorithm has converged after 8 iterations

On removing all misclassified points:
The Perceptron Algorithm has converged after 390 iterations

Linear SVM Dual
On removing all outliers:
The Perceptron Algorithm has converged after 10 iterations

On removing all misclassified points:
The Perceptron Algorithm has converged after 763 iterations
```

Primal Case

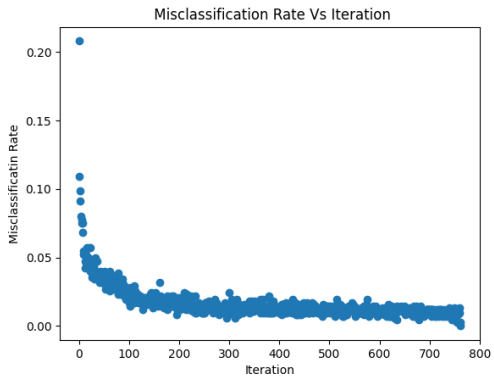


(a) (removed outliers)
Misclassification Rate vs Iteration

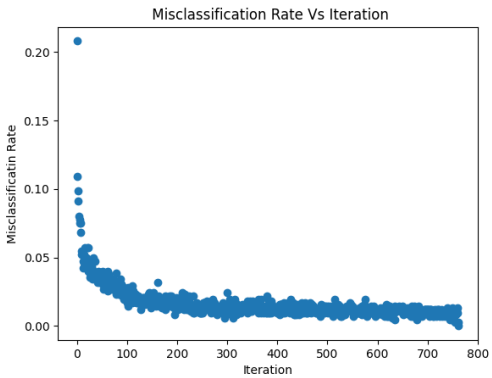


(b) (removed misclassified)
Misclassification Rate vs Iteration

Dual Case



(a) (removed outliers)
Misclassification Rate vs Iteration



(b) (removed misclassified)
Misclassification Rate vs Iteration

2 Logistic Regression, MLP, CNN & PCA

We are being given a subset of the MNIST-JPG dataset, where each image is grayscale and of size 28×28.
We load the dataset using Pytorch's ImageFolder and also normalize the data.

```
Dataset Size: 10000
Classes:  ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
Image Shape: torch.Size([1, 28, 28])
```

I then split the dataset into training and testing sets with a split ratio of 80-20.

```
length of Train Data: 8000
Lenght of Test Data: 2000
```

Task 1:Multi Layer Perceptron

For the implementation, I used the nn module of Pytorch.
The MLP consists of:

- Input Layer: 784 neurons (corresponding to flattened 28×28 images).
- Hidden Layer: 128 neurons with ReLU activation.
- Output Layer: 10 neurons (one for each class) with Softmax activation.

For the output layer, we use the Softmax activation to output the class probabilities.
The forward pass follows:

$$x = \text{ReLU}(W_1 x + b_1) \tag{10}$$

$$y = \text{Softmax}(W_2 x + b_2) \tag{11}$$

For training, I used:

- Loss Function:Negative Log-Likelihood Loss (`nn.NLLLoss()`).
- Optimizer: Stochastic Gradient Descent (SGD) with a learning rate of 0.001
- no of epochs: 10

On Test Dataset:

```
Accuracy of MLP:1504/2000 = 0.752
```

Let us randomly select an image and look at its prediction.

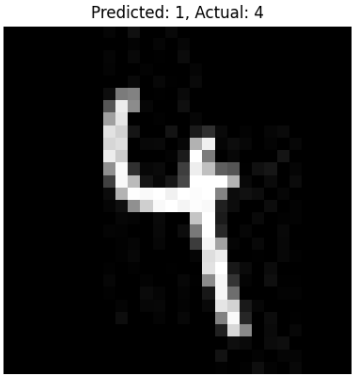


Figure 5: MLP Prediction

Task 2: Convolutional Neural Network

CNNs can directly take image as input and we want it to output the class probabilities.

I constructed the CNN as follows:

- Input Layer: 28×28 grayscale images (1 channel).
- Convolutional Layer 1: 32 filters, 3×3 kernel, stride 1, padding 1.
- ReLU Activation: Applies non-linearity.
- Max Pooling 1:2×2 pooling reduces feature map size to 14×14.
- Convolutional Layer 2: 64 filters, 3×3 kernel, stride 1, padding 1.
- ReLU Activation& Max Pooling 2:Reduces feature map size to 7×7.
- Fully Connected Layer 1:128 neurons.
- ReLU Activation.
- Output Layer:10 neurons (one for each class), followed by Softmax activation.

For training, I used:

- Loss Function: Negative Log-Likelihood Loss (`nn.NLLLoss()`).
- Optimizer: Adam optimizer with a learning rate of 0.001.
- no of epochs: 10

On test dataset:

```
Accuracy of CNN:1945/2000 = 0.9725
```

We see that CNN performs better than MLP.

Task 3: Principal Component Analysis

In this task we need to perform PCA on the dataset, which is a dimensionality reducing technique.
First we need to convert data to appropriate format, as I am implementing PCA using numpy.

```
Train Features Shape: (8000, 784)
Train Labels Shape: (8000,)
Test Features Shape: (2000, 784)
Test Labels Shape: (2000,)
```

After centering the training data, we calculate the covariance matrix and its eigenvalues and eigenvectors.

$$Cv = \lambda v \tag{12}$$

where:

- λ represents the eigenvalues.
- v represents the eigenvectors (principal component directions).

Then we sort the eigenvectors in decreasing order of eigenvalues and select the top $k = 200$ eigenvectors.
The transformed features in reduced dimensions are computed as:

$$X_{\text{PCA}} = X_{\text{centered}} V_k \tag{13}$$

where V_k contains the top k eigenvectors.

After PCA:

```
Eigen Vectors Shape: (784, 200)
Train Features PCA Shape: (8000, 200)
Test Features PCA Shape: (2000, 200)
```

Deliverable 1

Need to reconstruct any image of my choice using the top $k = 200$ principal components.

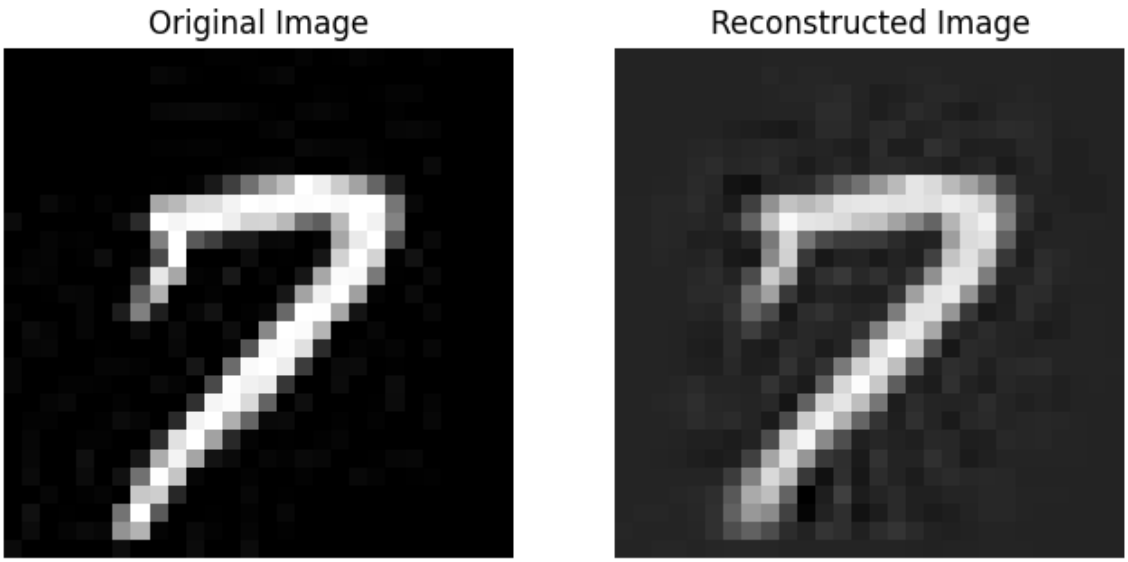


Figure 6: Original Image vs Reconstructed Image

Task 4: MLP with PCA

We now train the MLP on the PCA-transformed features.
Using the same architecture as before, we train the MLP on the PCA-transformed features.Firstly we need to convert the PCA-transformed features to the appropriate format for MLP, which is implemented using Pytorch.
Choosing:

- Loss Function: Negative Log-Likelihood Loss (`nn.NLLLoss()`).
- Optimizer: Stochastic Gradient Descent (SGD) with a learning rate of 0.001.
- no of epochs: 10

On test dataset:

```
Accuracy of MLP on PCA data:1414/2000 = 0.707
```

Task 5: Logistic Regression

We now train a Logistic Regression model on the PCA-transformed features.
For the implementation, I used numpy and:

- Softmax activation to compute class probabilities.
- Cross-entropy loss as the loss function.
- Gradient descent for optimization.
- no of epochs: 10

Logistic Regression (Multi-Class Classification):

```
Accuracy of Logistic Regression on PCA data: 0.7430
Misclassification Rate of Logistic Regression on PCA data: 0.2570
```

Now, we need to implement the "One Vs All" strategy in Logistic Regression.

- For each class, we train a separate binary classifier, so we have 10 classifiers in total.
- Each classifier treats one class as positive (+1) and all other classes as negative (0).
- As a decision rule, we choose the class with the highest probability.
- We train each classifier using the same PCA-transformed features.
- We now need to use Sigmod activation for the binary classification.
- We use the same cross-entropy loss function and Gradient Descent for optimization.
- no of epochs: 10 for each classifier.

Logistic Regression (One Vs All):

```
Accuracy of Logistic Regression OvR on PCA data: 0.7500
Misclassification Rate of Logistic Regression OvR on PCA data:
0.2500
```

For our dataset, the "One Vs All" strategy in Logistic Regression performs slightly better than the multi-class Logistic Regression.
In general, OvR approach is suitable when one class is dominant and the other classes are less frequent.

Deliverable 2

We need to print the confusion matrix for all multiclass classification models, and also report the metrics:
Accuracy, Precision, Recall, F1-Score for each class.
A confusion matrix is a N x N matrix, where N is the number of classes.
where:

$$C_{ij} = \text{Number of times a sample of class } i \text{ was predicted as class } j \tag{14}$$

Therefore,
C_{ii} (diagonal elements) represent correct classifications,
$C_{ij}, i \neq j$ (off-diagonal elements) represent misclassifications.
The metrics for each class are calculated as follows:

$$\text{Precision}_i = \frac{C_{ii}}{\sum_{j=0}^{N-1} C_{ji}} \tag{15}$$

$$\text{Recall}_i = \frac{C_{ii}}{\sum_{j=0}^{N-1} C_{ij}} \tag{16}$$

$$\text{F1-Score}_i = 2 \times \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{17}$$

$$\text{Per-Class Accuracy}_i = \frac{C_{ii}}{\sum_{j=0}^{N-1} C_{ij} + \sum_{j=0}^{N-1} C_{ji} - C_{ii}} \tag{18}$$

The Overall Metrics are computed as follows:

$$\text{Overall Precision} = \frac{1}{N} \sum_{i=0}^{N-1} \text{Precision}_i \tag{19}$$

$$\text{Overall Recall} = \frac{1}{N} \sum_{i=0}^{N-1} \text{Recall}_i \tag{20}$$

$$\text{Overall F1-Score} = \frac{1}{N} \sum_{i=0}^{N-1} \text{F1-Score}_i \tag{21}$$

$$\text{Overall Accuracy} = \frac{\sum_{i=0}^{N-1} C_{ii}}{\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C_{ij}} \tag{22}$$

For Precision, Recall and F-1 Score, we take the average of the per-class metrics.
But for Accuracy, we cannot take the average of per-class accuracies, as all the classes dont equally contribute to the accuracy. We might have some dominat classes and some rare classes, but average per-class accuracy would just treat all classes equally.

MLP of Task 1:

```
Confusion Matrix for MLP of Task1:

[[184   1   3   0   1   5   3   1   1   0]
 [  0 200   0   3   0   1   1   1   0   0]
 [  6  13 160   8   6   0   6   9   8   0]
 [  5  14   5 148   0   4   3   5   9   5]
 [  0   8   1   1 156   0   5   3   0  19]
 [ 15  35   0  19   6  95   5   1  16   7]
 [  4  16   5   0   3   2 170   0   0   0]
 [  2  17   5   0   1   0   0 169   0   6]
 [  2  19   8  12   1   2   3   1 145   8]
 [  5  15   1   4  22   3   0  17   0 121]]

Metrics for MLP
Overall Precision: 0.7894
Overall Recall: 0.7729
Overall F1 Score: 0.7701
Overall Accuracy: 0.7740

Per Class Metrics:
Class 0: Precision: 0.8251, Recall: 0.9246, F1 Score: 0.8720,
   Accuracy: 0.9730
Class 1: Precision: 0.5917, Recall: 0.9709, F1 Score: 0.7353,
   Accuracy: 0.9280
Class 2: Precision: 0.8511, Recall: 0.7407, F1 Score: 0.7921,
   Accuracy: 0.9580
Class 3: Precision: 0.7590, Recall: 0.7475, F1 Score: 0.7532,
   Accuracy: 0.9515
Class 4: Precision: 0.7959, Recall: 0.8083, F1 Score: 0.8021,
   Accuracy: 0.9615
Class 5: Precision: 0.8482, Recall: 0.4774, F1 Score: 0.6109,
   Accuracy: 0.9395
Class 6: Precision: 0.8673, Recall: 0.8500, F1 Score: 0.8586,
   Accuracy: 0.9720
Class 7: Precision: 0.8164, Recall: 0.8450, F1 Score: 0.8305,
   Accuracy: 0.9655
Class 8: Precision: 0.8101, Recall: 0.7214, F1 Score: 0.7632,
   Accuracy: 0.9550
Class 9: Precision: 0.7289, Recall: 0.6436, F1 Score: 0.6836,
   Accuracy: 0.9440
```

CNN of Task 2:

```
Confusion Matrix for CNN of Task2:
[[189   0   0   0   0   2   7   1   0   0]
 [  0 203   3   0   0   0   0   0   0   0]
 [  0   0 200   7   1   0   0   8   0   0]
 [  0   0   1 194   0   1   0   1   1   0]
 [  1   0   0   0 192   0   0   0   0   0]
 [  0   0   0   2   0 193   1   0   3   0]
 [  0   3   0   0   0   2 195   0   0   0]
 [  0   0   0   1   0   0   0 199   0   0]
 [  0   1   1   1   0   0   3   0 194   1]
 [  0   0   0   1   2   1   0   3   4 177]]

Metrics for CNN
Overall Precision: 0.9687
Overall Recall: 0.9682
Overall F1 Score: 0.9681
Overall Accuracy: 0.9680

Per Class Metrics:
Class 0: Precision: 0.9947, Recall: 0.9497, F1 Score: 0.9717,
   Accuracy: 0.9945
Class 1: Precision: 0.9807, Recall: 0.9854, F1 Score: 0.9831,
   Accuracy: 0.9965
Class 2: Precision: 0.9756, Recall: 0.9259, F1 Score: 0.9501,
   Accuracy: 0.9895
Class 3: Precision: 0.9417, Recall: 0.9798, F1 Score: 0.9604,
   Accuracy: 0.9920
Class 4: Precision: 0.9846, Recall: 0.9948, F1 Score: 0.9897,
   Accuracy: 0.9980
Class 5: Precision: 0.9698, Recall: 0.9698, F1 Score: 0.9698,
   Accuracy: 0.9940
Class 6: Precision: 0.9466, Recall: 0.9750, F1 Score: 0.9606,
   Accuracy: 0.9920
Class 7: Precision: 0.9387, Recall: 0.9950, F1 Score: 0.9660,
   Accuracy: 0.9930
Class 8: Precision: 0.9604, Recall: 0.9652, F1 Score: 0.9628,
   Accuracy: 0.9925
Class 9: Precision: 0.9944, Recall: 0.9415, F1 Score: 0.9672,
   Accuracy: 0.9940
```

Task3 was PCA, so no confusion matrix

MLP with PCA of Task 4:

```
Confusion Matrix for MLP on PCA data of Task4:

[[185   0   0   0   1   0  11   2   0   0]
 [  0 201   1   0   0   0   2   1   1   0]
 [ 13  28 125  16   2   0  19   5   7   1]
 [  8  14   6 144   1   6   2   8   8   1]
 [  5   9   1   1 147   1   9   4   2  14]
 [ 32  31   0  22  14  65  16   4   7   8]
 [  3  14   2   0   2   1 176   0   2   0]
 [  5  14   1   1   4   0   0 170   1   4]
 [  6  27   4  23   2   7   7   6 113   6]
 [  7  20   0   3  31   2   0  30   1  94]]

Metrics for MLP on PCA Data
Overall Precision: 0.7351
Overall Recall: 0.7092
Overall F1 Score: 0.6964
Overall Accuracy: 0.7100

Per Class Metrics:
Class 0: Precision: 0.7008, Recall: 0.9296, F1 Score: 0.7991,
   Accuracy: 0.9535
Class 1: Precision: 0.5615, Recall: 0.9757, F1 Score: 0.7128,
   Accuracy: 0.9190
Class 2: Precision: 0.8929, Recall: 0.5787, F1 Score: 0.7022,
   Accuracy: 0.9470
Class 3: Precision: 0.6857, Recall: 0.7273, F1 Score: 0.7059,
   Accuracy: 0.9400
Class 4: Precision: 0.7206, Recall: 0.7617, F1 Score: 0.7406,
   Accuracy: 0.9485
Class 5: Precision: 0.7927, Recall: 0.3266, F1 Score: 0.4626,
   Accuracy: 0.9245
Class 6: Precision: 0.7273, Recall: 0.8800, F1 Score: 0.7964,
   Accuracy: 0.9550
Class 7: Precision: 0.7391, Recall: 0.8500, F1 Score: 0.7907,
   Accuracy: 0.9550
Class 8: Precision: 0.7958, Recall: 0.5622, F1 Score: 0.6589,
   Accuracy: 0.9415
Class 9: Precision: 0.7344, Recall: 0.5000, F1 Score: 0.5949,
   Accuracy: 0.9360
```

Logistic Regression of Task 5:

```
Confusion Matrix for Logistic Regression on PCA data of Task5:

[[193   0   0   1   1   0   3   1   0   0]
 [  0 204   0   0   0   0   1   1   0   0]
 [ 15  27 133  14   7   0   8   6   6   0]
 [ 11  23   2 150   0   1   1   6   4   0]
 [  3  11   0   0 158   0   5   4   1  11]
 [ 45  39   0  40   6  54   7   4   1   3]
 [  5  21   2   0   0   1 171   0   0   0]
 [  4  21   2   0   2   0   0 170   0   1]
 [  6  37   2  15   3   1   5   4 123   5]
 [  6  19   1   4  26   1   0  21   0 110]]

Metrics for Logistic Regression on PCA Data
Overall Precision: 0.7885
Overall Recall: 0.7326
Overall F1 Score: 0.7228
Overall Accuracy: 0.7330

Per Class Metrics:
Class 0: Precision: 0.6701, Recall: 0.9698, F1 Score: 0.7926,
   Accuracy: 0.9495
Class 1: Precision: 0.5075, Recall: 0.9903, F1 Score: 0.6711,
   Accuracy: 0.9000
Class 2: Precision: 0.9366, Recall: 0.6157, F1 Score: 0.7430,
   Accuracy: 0.9540
Class 3: Precision: 0.6696, Recall: 0.7576, F1 Score: 0.7109,
   Accuracy: 0.9390
Class 4: Precision: 0.7783, Recall: 0.8187, F1 Score: 0.7980,
   Accuracy: 0.9600
Class 5: Precision: 0.9310, Recall: 0.2714, F1 Score: 0.4202,
   Accuracy: 0.9255
Class 6: Precision: 0.8507, Recall: 0.8550, F1 Score: 0.8529,
   Accuracy: 0.9705
Class 7: Precision: 0.7834, Recall: 0.8500, F1 Score: 0.8153,
   Accuracy: 0.9615
Class 8: Precision: 0.9111, Recall: 0.6119, F1 Score: 0.7321,
   Accuracy: 0.9550
Class 9: Precision: 0.8462, Recall: 0.5851, F1 Score: 0.6918,
   Accuracy: 0.9510
```

Logistic Regression OvR of Task 5:

```
Confusion Matrix for Logistic Regression OvR on PCA data of Task5
[[193   0   0   1   1   0   3   1   0   0]
[  0 204   0   0   0   0   1   1   0   0]
[ 15  27 133  14   7   0   8   6   6   0]
[ 11  22   2 151   0   1   1   6   4   0]
[  3  11   0   0 158   0   5   4   1  11]
[ 45  39   0  40   6  54   7   4   1   3]
[  5  20   2   0   0   1 172   0   0   0]
[  4  21   2   0   2   0   0 170   0   1]
[  6  36   2  15   3   1   5   4 124   5]
[  6  19   1   4  26   1   0  21   0 110]]

Metrics for Logistic Regression OvR on PCA Data
Overall Precision: 0.7891
Overall Recall: 0.7341
Overall F1 Score: 0.7241
Overall Accuracy: 0.7345

Per Class Metrics:
Class 0: Precision: 0.6701, Recall: 0.9698, F1 Score: 0.7926,
   Accuracy: 0.9495
Class 1: Precision: 0.5113, Recall: 0.9903, F1 Score: 0.6744,
   Accuracy: 0.9015
Class 2: Precision: 0.9366, Recall: 0.6157, F1 Score: 0.7430,
   Accuracy: 0.9540
Class 3: Precision: 0.6711, Recall: 0.7626, F1 Score: 0.7139,
   Accuracy: 0.9395
Class 4: Precision: 0.7783, Recall: 0.8187, F1 Score: 0.7980,
   Accuracy: 0.9600
Class 5: Precision: 0.9310, Recall: 0.2714, F1 Score: 0.4202,
   Accuracy: 0.9255
Class 6: Precision: 0.8515, Recall: 0.8600, F1 Score: 0.8557,
   Accuracy: 0.9710
Class 7: Precision: 0.7834, Recall: 0.8500, F1 Score: 0.8153,
   Accuracy: 0.9615
Class 8: Precision: 0.9118, Recall: 0.6169, F1 Score: 0.7359,
   Accuracy: 0.9555
Class 9: Precision: 0.8462, Recall: 0.5851, F1 Score: 0.6918,
   Accuracy: 0.9510
```

Observations:

- CNN outperforms all models in terms of accuracy. It has high precision and recall across all classes.
- Models trained on PCA-transformed features show a drop in performance (accuracy) compared to the models performing on raw data. This is expected as PCA is a dimensionality reduction technique and some information is lost in the process.So basically it introduces a tradeoff between computation complexity and accuracy.
- We see that classes 0 and 1 have high recalls accorss all trained models. This means that the models are able to correctly classify these classes with a better rate.
- Class 5 has the lowest recall across all models suggesting that a significant amount of sample of this class are being misclassified.

- The "One Vs All" strategy in Logistic Regression performs slightly better than the multi-class Logistic Regression. This is because the dataset is imbalanced and the OvR approach is suitable when one class is dominant and the other classes are less frequent.
- Increasing the Model Complexity (More layers etc) can help in improving the performance of the models, as we can see CNN and MLP perform better than Logistic Regression.

Deliverable 3

We need to compute the AUC scores and plot the ROC curves for each class obtained from the Logistic Regression OvR model.
ROC Curve: Reciever Operating Characteristic Curve is a plot of the True Positive Rate (Recall) against the False Positive Rate at different classification thresholds.

- **True Positive Rate (TPR) or Recall:** Measures how many actual positives were correctly classified.
$$TPR = \frac{TP}{TP + FN} \tag{23}$$
- **False Positive Rate (FPR):** Measures how many actual negatives were incorrectly classified as positives.
$$FPR = \frac{FP}{FP + TN} \tag{24}$$

AUC Score: Area under the ROC Curve quantifies the performance of the model in distinguishing between different classes.

```
Class 0: AUC Score: 0.9865
Class 1: AUC Score: 0.9897
Class 2: AUC Score: 0.9528
Class 3: AUC Score: 0.9471
Class 4: AUC Score: 0.9663
Class 5: AUC Score: 0.9122
Class 6: AUC Score: 0.9770
Class 7: AUC Score: 0.9807
Class 8: AUC Score: 0.9411
Class 9: AUC Score: 0.9175

Average AUC Score: 0.9571
```

Following are the ROC Curves for each class:

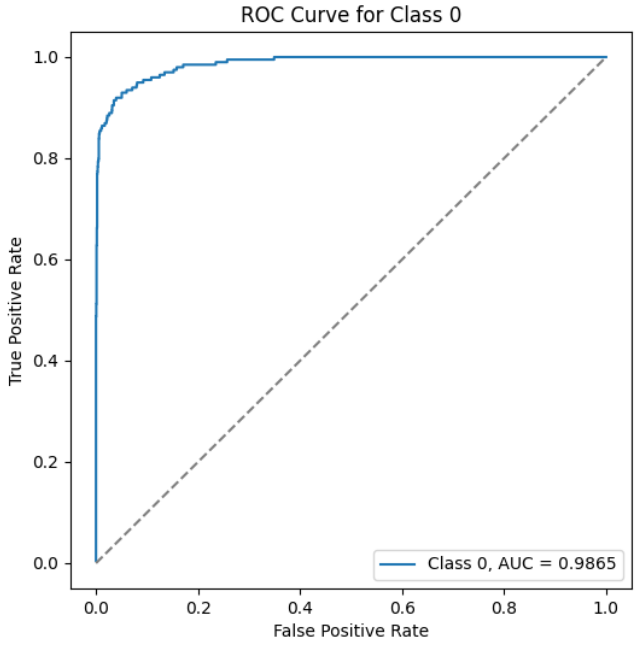


Figure 7: ROC Curves for class 0

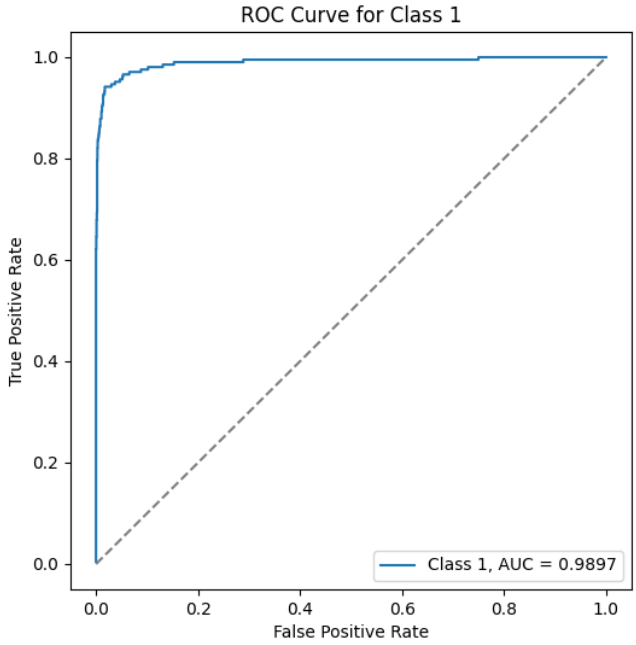


Figure 8: ROC Curves for class 1

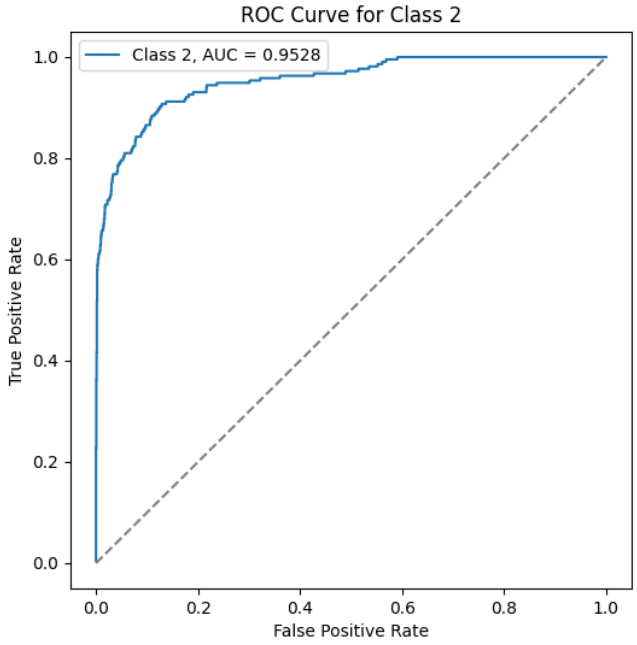


Figure 9: ROC Curves for class 2

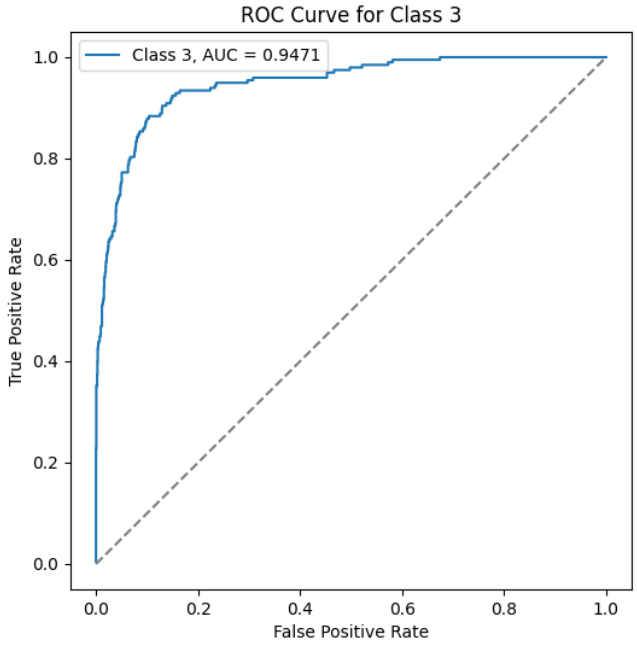


Figure 10: ROC Curves for class 3

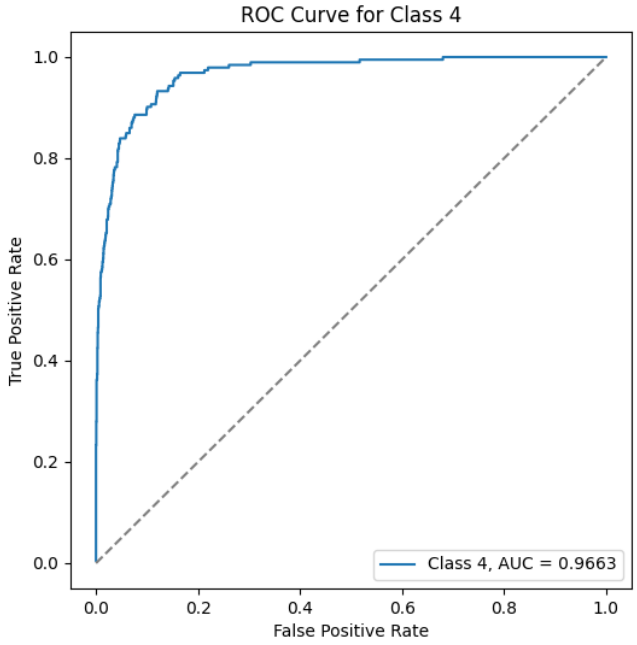


Figure 11: ROC Curves for class 4

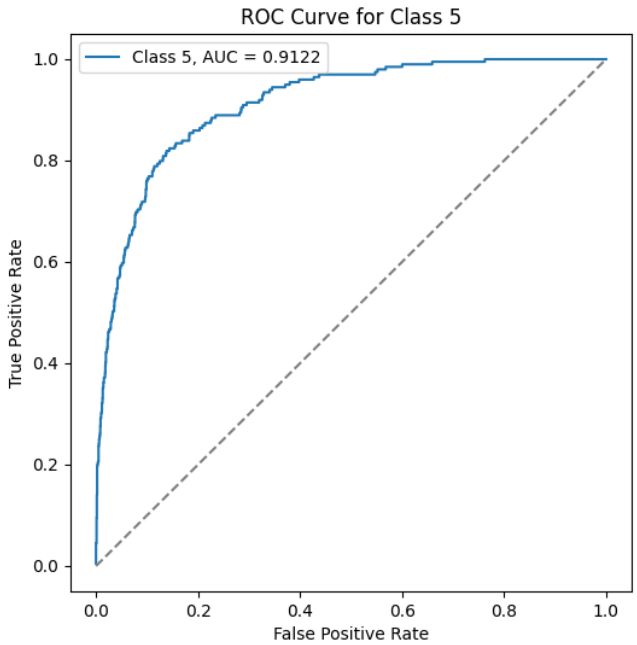


Figure 12: ROC Curves for class 5

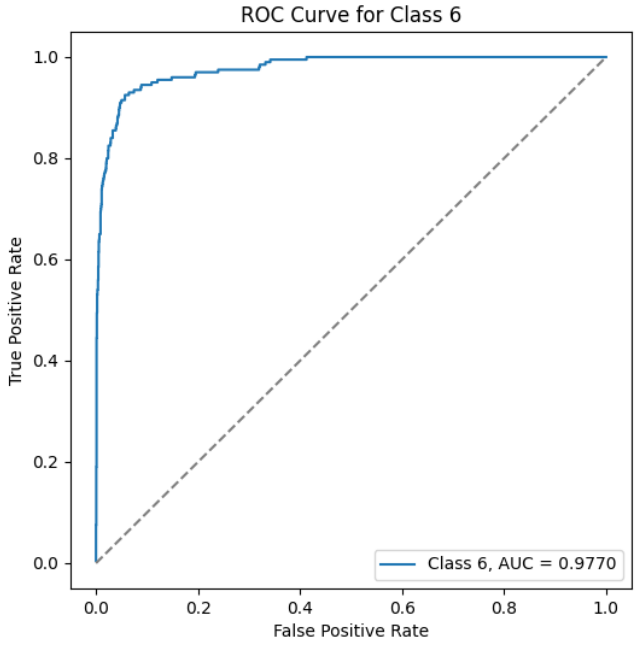


Figure 13: ROC Curves for class 6

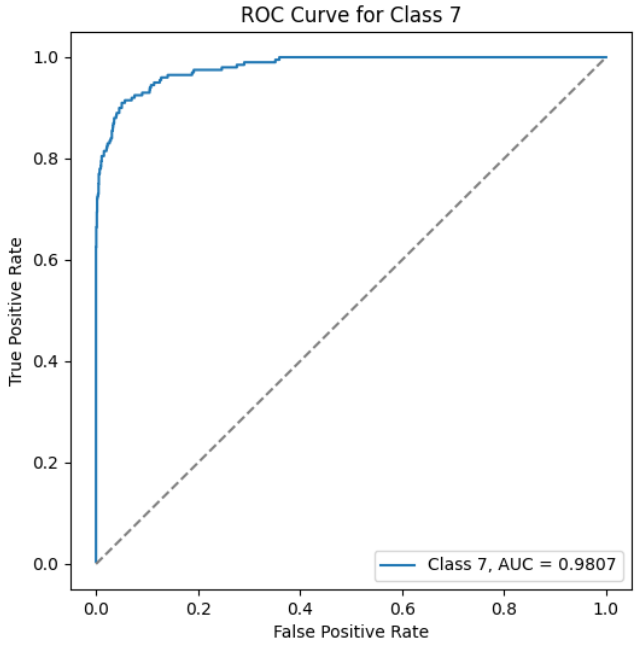


Figure 14: ROC Curves for class 7

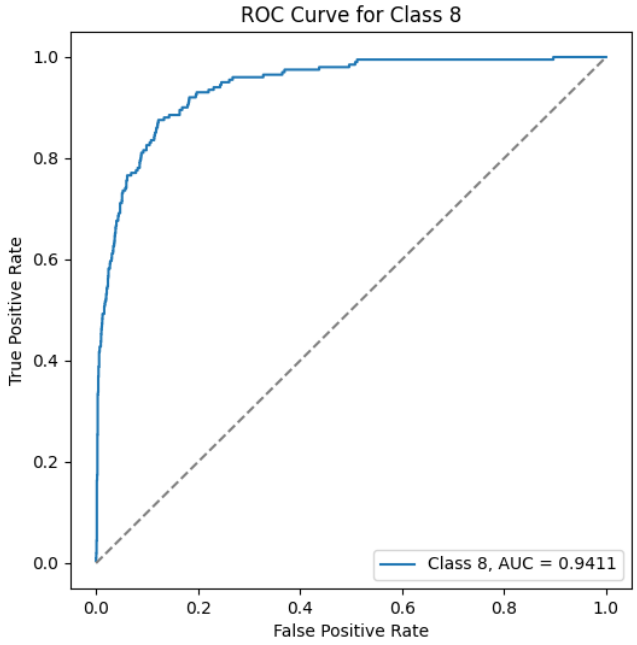


Figure 15: ROC Curves for class 8

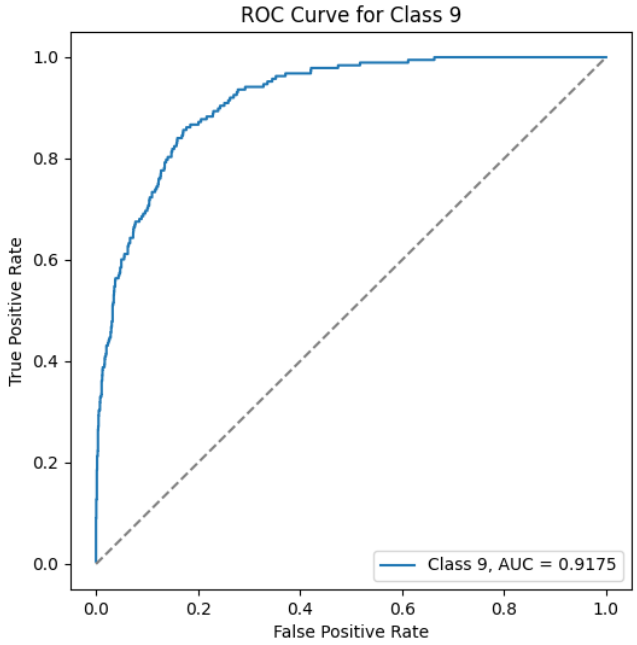


Figure 16: ROC Curves for class 9

3 Regression

3.1 Linear Regression

We are required to solve a regression problem using two approaches:

- **Ordinary Least Squares (OLS)**, which minimizes the objective function:

$$J(w) = \frac{1}{2n}\sum_{i=1}^{m}(y^{(i)} - w^T x^{(i)})^2 = \frac{1}{2n}||Y - Xw||^2, \quad (X \in \mathbb{R}^{n\times d}, Y \in \mathbb{R}^n) \tag{25}$$

- **Ridge Regression (RR)**, which includes an additional penalty term:

$$J(w) = \frac{1}{2n}||Y - Xw||^2 + \frac{\lambda}{2}||w||^2 \tag{26}$$

Task 1: Obtain Data

We query the oracle to obtain two datasets, each having training and test data, i.e., $\mathcal{D}_1^{train}, \mathcal{D}_1^{test}$ and $\mathcal{D}_2^{train}, \mathcal{D}_2^{test}$.
The datasets obtained are not numpy arrays, so we first need to convert them to appropriate format.

```
Dataset 1
Shape of X train (20, 5)
shape of y train (20, 1)

Dataset 2
Shape of X train (40, 100)
shape of y train (40, 1)
```

Task 2: Solve OLS and RR for both datasets

Ordinary Least Squares (OLS):
For OLS, we need to estimate the weight vector w using the closed-form solution:

$$w_{\text{OLS}} = (X^T X)^{-1} X^T Y \tag{27}$$

where:

- $X \in \mathbb{R}^{n\times d}$ is the feature matrix.
- $Y \in \mathbb{R}^n$ is the target vector.
- $w \in \mathbb{R}^d$ is the weight vector.

Ridge Regression (RR):
Ridge regression modifies the OLS solution by introducing a regularization term:

$$w_{\text{RR}} = (X^T X + \lambda I)^{-1} X^T Y \tag{28}$$

where:

- λ is the regularization parameter.
- I is the identity matrix.

We will take $\lambda = 1$ for Ridge Regression.
Let:

$\mathbf{w}_1^{ols}, \mathbf{w}_1^{rr}$ be the weights for dataset 1, corresponding to OLS and RR.

$\mathbf{w}_2^{ols}, \mathbf{w}_2^{rr}$ be the weights for dataset 2, corresponding to OLS and RR.

```
Weights for Dataset 1 using OLS:
Shape of Weights:  (6, 1)

Weights for Dataset 2 using OLS:
Shape of Weights:  (101, 1)

Weights for Dataset 1 using Ridge Regression:
Shape of Weights:  (6, 1)

Weights for Dataset 2 using Ridge Regression:
Shape of Weights:  (101, 1)
```

Task 3: Calculate MSE for w_1^{ols}, w_1^{rr} using $\mathcal{D}_1^{test}$ and w_2^{ols}, w_2^{rr} using $\mathcal{D}_2^{test}$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda||w||^2 \tag{29}$$

when using OLS, we take $\lambda = 0$.
when using RR, we take $\lambda = 1$.

Deliverable 1

Question: Can we do OLS if X (Feature Matrix) doesn't have full rank?

Answer: No. The feature matrix needs to have a full rank in order to do Ordinary Least Squares (OLS).
This is because, in OLS we need to calculate the inverse of X^TX,
as $w_{\text{OLS}} = (X^TX)^{-1}X^TY$.
If X is not full rank, then X^TX will not be invertible.

If X^TX is singular then we get undefined or unstable solutions.

In this case, we can use the pseudo inverse to solve the OLS problem.
We can also use the ridge regression, as it adds a regularization term to the OLS problem, thus making the solution stable.

```
MSE for Dataset 2 using OLS:
MSE: 160290.2198 (Unstable)

Rank of X_train_2:  40
Number of features in X_train_2:  100
Since rank is less than number of features, X_train_2 doesnt have
   full rank
Determinant of X.T*X:  -0.0
Rank of X_train_1:  5
Number of features in X_train_1:  5
Since rank is equal to number of features, X_train_1 has full rank
so, OLS can be applied to Dataset 1
MSE for Dataset 1 using OLS:
MSE: 0.0656

MSE for Dataset 2 using OLS: (Using Pseudo Inverse)
MSE: 20.3753
MSE for Dataset 2 using Ridge Regression:
MSE: 32.5397
```

Deliverable 2

Need to report MSE on $\mathcal{D}_1^{\text{test}}$ and corresponding weights for OLS and RR.

```
MSE for Dataset 1:
Using OLS:
MSE on train data: 0.0259
MSE on test data: 0.0656

Using Ridge Regression: (with added penalty in cost function)
MSE on train data: 0.5257
MSE on test data: 0.5590

Weights and bias of w_1_ols:
[0.49725451 0.2445854  0.15961118 0.07685568 0.44338183]
0.0007828173830240481

Weights and bias of w_1_rr:
[0.48061895 0.22471611 0.16152503 0.06763827 0.43204286]
-0.0006882139273898802
```

Deliverable 3

Need to report MSE on $\mathcal{D}_2^{\text{test}}$ and corresponding weights for OLS and RR.

```
MSE for Dataset 2:
Using OLS: (with Pseudo Inverse)
MSE on train data: 0.0000
MSE on test data: 20.375

Using Ridge Regression: (with added penalty in cost function)
MSE on train data: 12.0750
MSE on test data: 32.5397
```

Saving the weights and bias of the models in a file.
Attached: 1.**w_ols_23651.csv** 2.**w_rr_23652.csv**

3.2 Support Vector Regression

We will use the SVR model on our dataset using both Linear and Gaussian (RBF) kernels.
The dataset consists of historical stock prices, from which we extract the closing prices, normalize them, and construct X and y using time window t.

Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i represents the historical stock prices over t days and y_i represents the stock price on the next day, our goal is to learn a function $f(x)$ that minimizes the ϵ-insensitive loss function:

$$E_\epsilon(f(x) - y) = \begin{cases} 0, & |f(x) - y| < \epsilon \\ |f(x) - y| - \epsilon, & \text{otherwise} \end{cases} \tag{30}$$

The optimization problem for SVR is formulated as:

$$\min_{w,b} \quad C \sum_{n=1}^{N} E_\epsilon(f(x_n) - y_n) + \frac{1}{2}||w||^2 \tag{31}$$

where $f(x) = w^T \phi(x) + b$, and $\phi(x)$ represents a feature space transformation.
Data Preprocessing: Assigned Stock: AGFS

1. Extract closing prices from the CSV file, call this vector d.
2. Normalize the data using StandardScaler:

$$x' = \frac{x - \mu}{\sigma} \tag{32}$$

3. Construct a dataset matrix X where each row contains the closing prices of the past t days.
4. Construct a target vector y where each element is the closing price of the next day.
5. We split the data into training and testing sets with split rate 50-50.
6. We train the SVR model using $C = 10$ and $\epsilon = 1e - 3$.

```
Shape of d: (701, 1)

Shape of X: (694, 7)
Shape of y: (694,)

Mean of normalized data: 4.054452129872184e-17
Standard deviation of normalized data: 1.0
Time Window: 7

Training with C=10, epsilon=0.001
```

Task 1: Linear SVR using Dual formalism

We will use the dual formalism of the kernelized SVR model with a linear Kernel.

$$\max_{\alpha,\alpha^*} -\frac{1}{2}(\alpha - \alpha^*)^T K(\alpha - \alpha^*) - \epsilon \sum(\alpha + \alpha^*) + y^T(\alpha - \alpha^*) \tag{33}$$

subject to:

$$0 \leq \alpha_i, \alpha_i^* \leq C, \quad \sum(\alpha_i - \alpha_i^*) = 0 \tag{34}$$

The kernel matrix K for the linear case is computed as:

$$K = XX^T \tag{35}$$

We trained models for different time windows $t \in \{7, 30, 90\}$:

```
Time Window: 7
Number of support vectors: 343
Weight Vector (w):
[ 0.0320884  -0.03248221  0.05233977  0.03889653 -0.14409022
   0.03470537 1.01695284]
Bias (b): -0.0027083029638147256
Loss: 0.04858729484265818

Time Window: 30
Number of support vectors: 320
Weight Vector (w):
[ 3.69957676e-02  1.77834625e-02 -2.54470455e-02  6.00188323e-02
   -9.83826286e-02  6.74847826e-02 -4.00263412e-02 -1.18962272e-01
   8.40328331e-02  2.94207738e-02 -1.92217153e-04 -3.56446927e-02
   -3.32200467e-02  6.90346912e-02 -1.11865933e-01 -1.19799491e-02
   1.43971629e-01 -2.40747537e-02 -8.22981410e-02  1.27071464e-02
   3.94749386e-02  8.12146881e-02 -6.00166610e-02  4.27833587e-02
   -4.93581399e-02  6.58276683e-02  8.66112981e-02 -2.04018446e-01
   -3.03094215e-02  1.08675628e+00]
Bias (b): -0.0027411326638789734
Loss: 0.05393047950161548

Time Window: 90
Number of support vectors: 260
Weight Vector (w):
[-3.11365642e-02  1.53816285e-01 -1.59441057e-02 -1.32943096e-01
   -2.38357061e-05  1.11897778e-01 -1.87274280e-02 -8.99823036e-02
   -6.39649756e-02  9.16713981e-02  2.31036859e-02 -1.37580602e-01
   3.20388688e-02  1.71428456e-01 -1.17066307e-01 -1.01697980e-01
   4.75272435e-02  1.21716428e-01 -9.45571187e-03  3.64734542e-03
   -5.55470026e-02  1.35076391e-02  4.02738794e-02 -1.31188812e-01
   4.22874325e-03  7.56241552e-02  5.81320437e-02 -6.91942517e-02
   1.20757001e-02 -4.98650167e-02  6.88095306e-02  3.95213132e-02
   -1.52902280e-01  3.73920346e-02  9.66918377e-03 -3.65054036e-02
   1.14888098e-01 -1.32506159e-01  6.38085559e-02  2.03766679e-02
   4.31545519e-02 -2.06702821e-01  1.95490371e-01  6.94338304e-03
   -3.08304888e-02  1.23328478e-03  2.56339137e-02 -5.43410988e-02
   1.25843904e-02 -2.18077385e-02  6.25323873e-02  5.30079099e-02
   -1.81923459e-02 -1.24749879e-01  9.62846016e-02 -8.23102888e-02
   -1.35200810e-02  1.38531592e-01 -5.09754949e-02  7.05326770e-02
   1.43065316e-02 -5.78905185e-02  1.29672078e-03  6.21207350e-02
```

```
    -1.14455738e-01  1.37126488e-01 -9.04282924e-02 -9.39686084e-02
    2.69488224e-02  8.22526756e-02  3.44555731e-02 -1.10414773e-01
    3.41268148e-02  3.51509946e-02 -7.34224079e-02 -5.80464172e-02
    1.04707319e-01  5.15200997e-02 -9.28370184e-02  2.45239437e-02
    -3.13208400e-02  1.13129588e-01 -5.55109462e-02 -4.12758716e-02
    -1.13050638e-02  7.98904925e-02  8.20035531e-02 -2.03678417e-01
    1.73951099e-03  1.06524944e+00]
Bias (b): 0.007396808487452 43
Loss: 0.06661065822164929
```

Task 2: Gaussian Kernel SVR

For nonlinear regression, we used the Gaussian RBF kernel:

$$K(x_i, x_j) = \exp\left(-\gamma||x_i - x_j||^2\right) \tag{36}$$

where γ is the kernel parameter.
We trained models for different time windows $t \in \{7, 30, 90\}$, and for each time window, we varied γ in {1, 0.1, 0.01, 0.001}.

```
Time Window: 7
Number of support vectors: 343

>>>Gamma: 1
Loss: 0.6457195145310546
>>>Gamma: 0.1
Loss: 0.1685556352338386
>>>Gamma: 0.01
Loss: 0.07810958598906532
>>>Gamma: 0.001
Loss: 0.06209339237886469

Time Window: 30
Number of support vectors: 320

>>>Gamma: 1
Loss: 0.8144236650130938
>>>Gamma: 0.1
Loss: 0.6873127404028401
>>>Gamma: 0.01
Loss: 0.24214894841599277
>>>Gamma: 0.001
Loss: 0.10819812710952408

Time Window: 90
Number of support vectors: 260

>>>Gamma: 1
Loss: 1.0206596471422342
>>>Gamma: 0.1
Loss: 0.9444052425133608
>>>Gamma: 0.01
Loss: 0.8337357580683322
>>>Gamma: 0.001
Loss: 0.23164247827628542
```

We see that the loss decreases as we decrease the value of γ.
And the loss increases as we increase the time window.

Deliverables:

For each SVR trained we need to plot a graph on the test set containing:

- Predicted closing price values.
- True closing price values.
- Average price on previous days.

For Linear SVR:

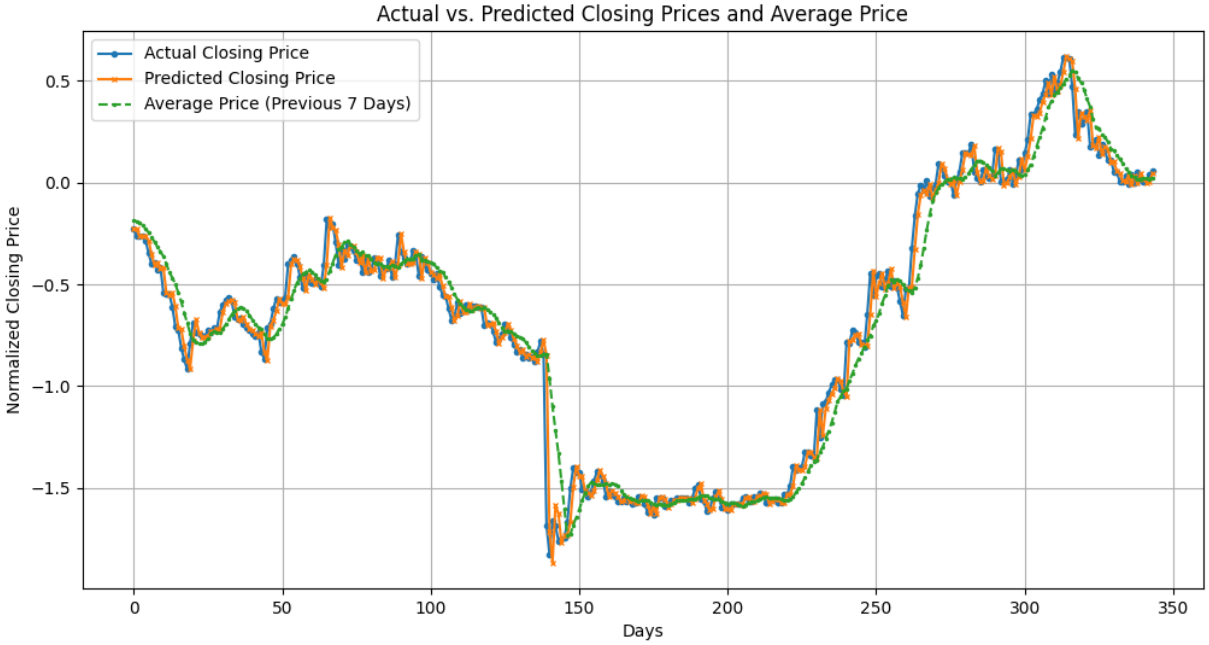


Figure 17: Linear SVR for Time Window 7

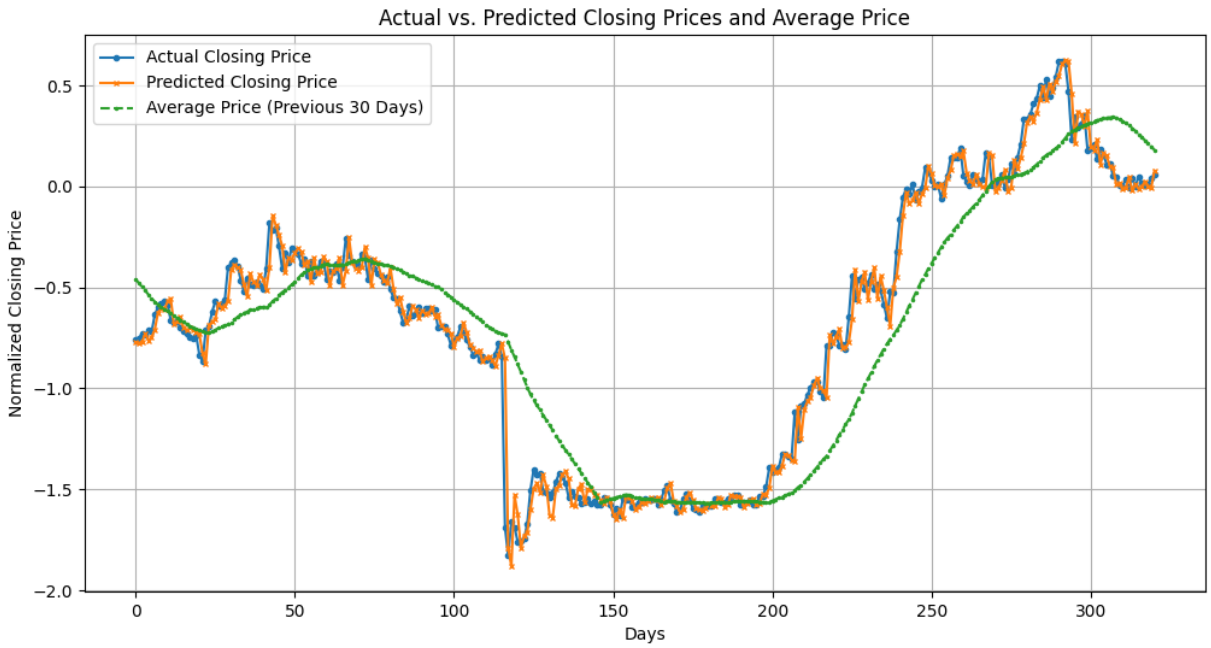


Figure 18: Linear SVR for Time Window 30

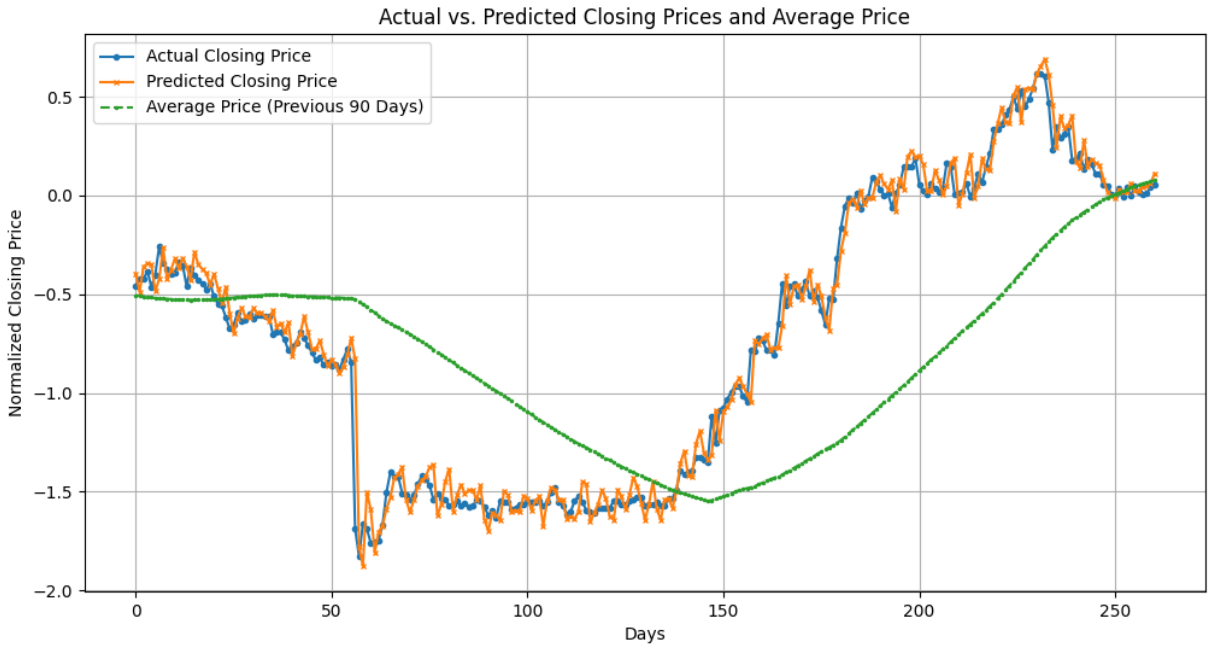


Figure 19: Linear SVR for Time Window 90

For Gaussian Kernel SVR:

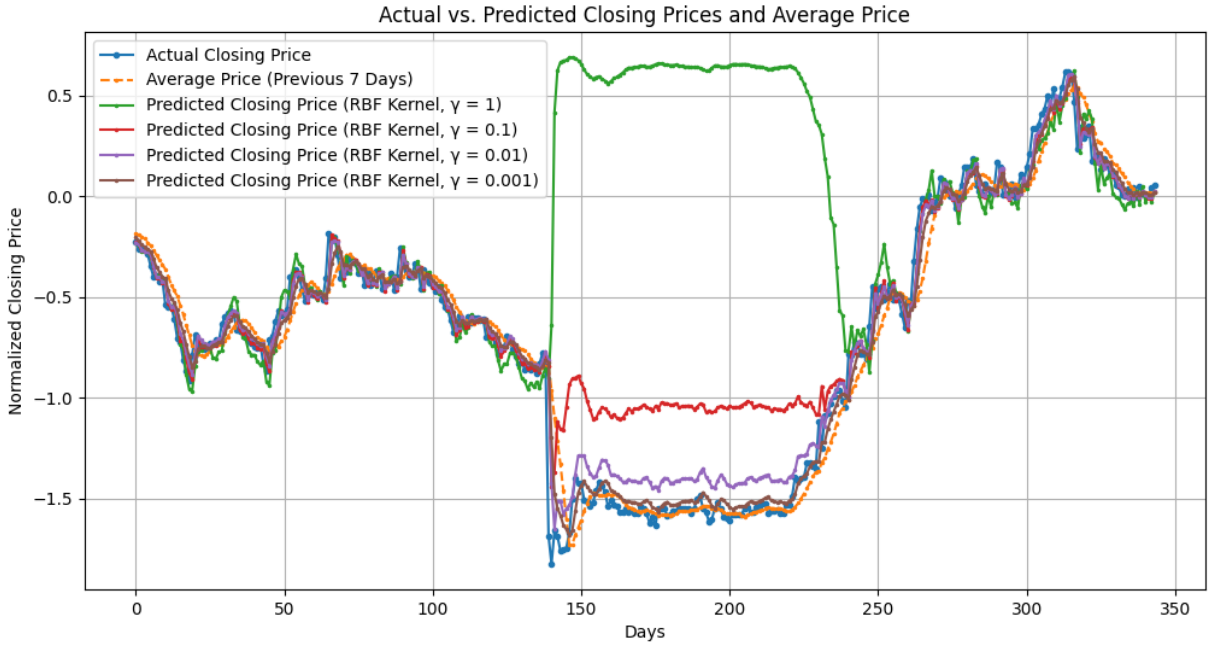


Figure 20: Gaussian Kernel SVR for Time Window 7

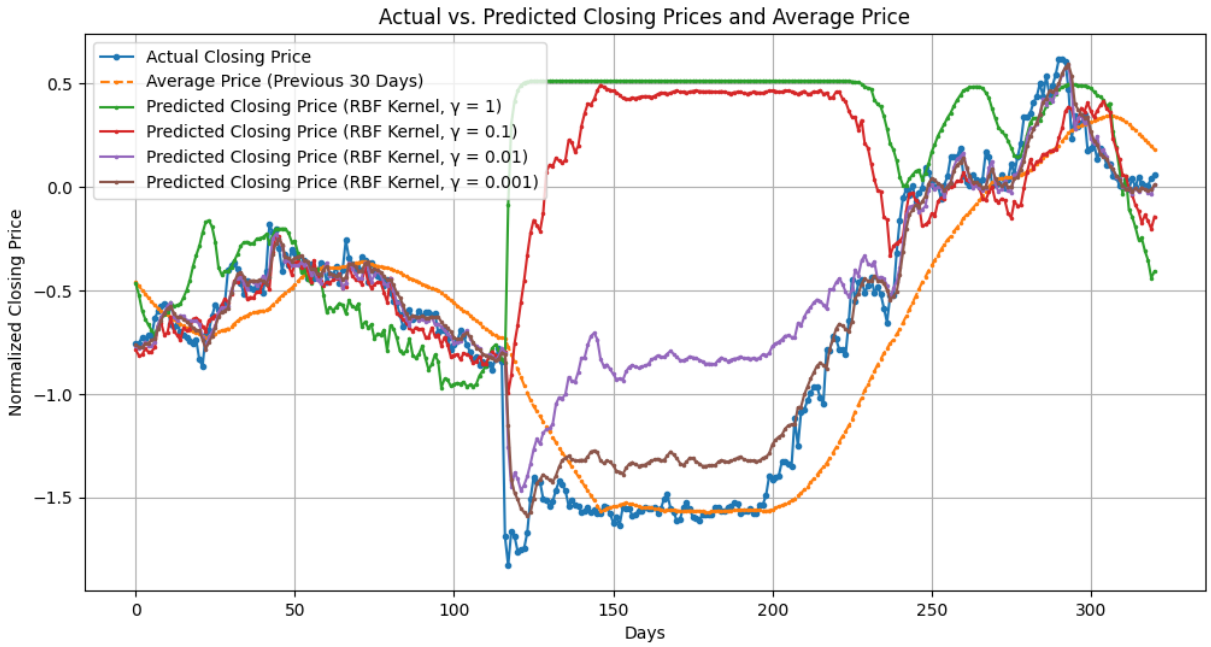


Figure 21: Gaussian Kernel SVR for Time Window 30

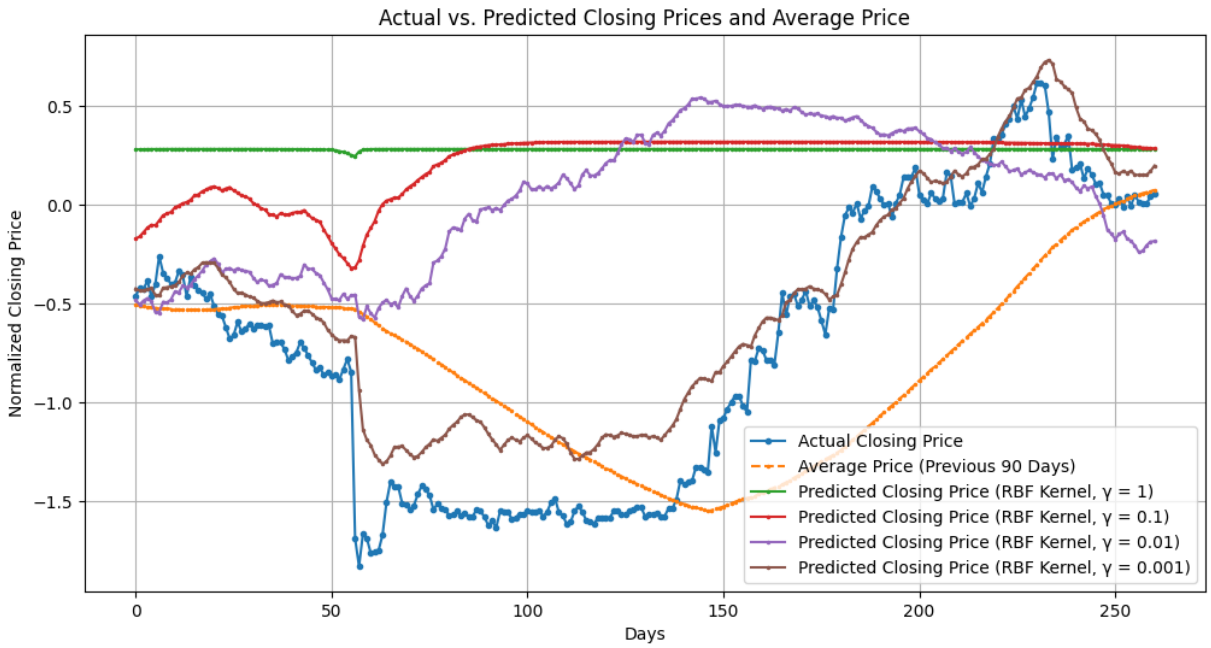


Figure 22: Gaussian Kernel SVR for Time Window 90