# UMC203: Artificial Intelligence and Machine Learning

Assignment 2

Kintan Saha SR No. 23881

July 5, 2025

# Problem 1:

## Task 1: Perceptron

The perceptron algorithm is run on the training data with the number of iterations capped at $10^9$. The perceptron doesn't converge after such a large number of iterations and as such we can safely conclude that the perceptron won't converge. The number of misclassifications on the test dataset of the perceptron algorithm with the iterations is given in Figure 1. The misclassification rate can found in Figure 2.

## Task 2: Linear slack SVM

The primal and the dual of the linear slack SVM is solved using `cvxopt`. In general, it is observed that the dual is solved slightly slower than the primal.
Time taken by dual to solve the linear slack SVM : $0.6041646003723145s$
Time taken by primal to solve the linear slack SVM: $0.5674755573272705s$
The SVM solution is used to identify the images responsible for misclassification.

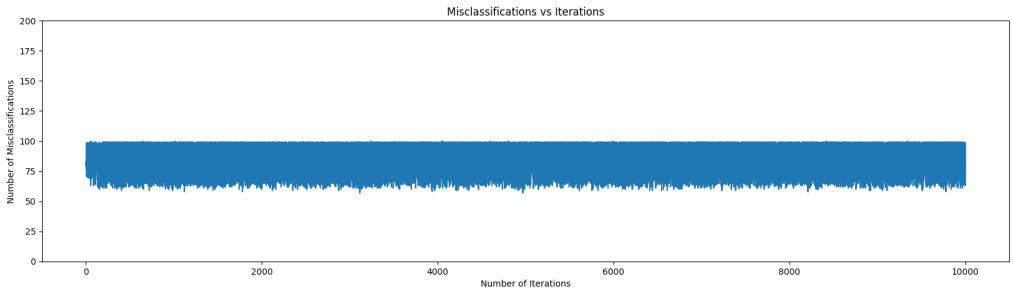


Figure 1: number of misclassifications for perceptron with iterations

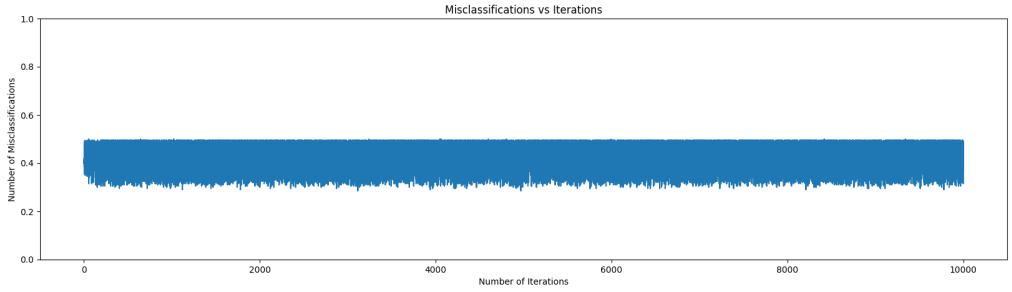


Figure 2: Misclassification rate of Perceptron with iterations

## Task 3: Images that cause non-separability

The images that cause non-separability can be identified using the solution for the SVM. Such images $(x^{(i)}, y^{(i)})$ are either misclassified by the SVM or are inside the margin. All such images will make the Perceptron to not converge. Formally such images satisfy the following condition: $y^{(i)}(w_{SVM}^T x^{(i)} + b) < 1$

Testing all the images in `train_images` using this condition we get the following 773 ( 318 are images which got misclassified and 455 are images which were inside the margin) indices of the images which cause non-separability:

```
[6, 24, 28, 30, 33, 37, 38, 39, 42, 44, 46, 51, 57, 61, 69, 70, 73, 75, 81, 83,
89, 90, 93, 94, 104, 105, 106, 108, 113, 114, 119, 122, 123, 128, 137, 147, 151,
152, 153, 156, 162, 163, 164, 165, 169, 170, 174, 176, 178, 183, 185, 187, 188,
189, 193, 196, 197, 199, 203, 204, 205, 207, 208, 210, 223, 224, 229, 230, 233,
237, 238, 240, 243, 252, 253, 258, 260, 263, 264, 266, 272, 274, 278, 279, 282,
283, 284, 289, 293, 294, 296, 297, 298, 301, 302, 304, 305, 306, 307, 308, 309,
310, 311, 315, 318, 319, 323, 326, 327, 337, 340, 341, 342, 343, 352, 361, 363,
369, 372, 376, 380, 389, 394, 396, 397, 399, 401, 403, 405, 409, 410, 414, 418,
419, 429, 431, 436, 442, 448, 449, 451, 457, 460, 461, 465, 468, 470, 471, 483,
484, 488, 490, 494, 495, 498, 501, 503, 504, 505, 509, 510, 517, 518, 534, 536,
538, 541, 542, 555, 562, 563, 564, 565, 567, 569, 571, 572, 579, 583, 586, 592,
595, 602, 607, 611, 614, 616, 619, 620, 621, 626, 630, 634, 638, 639, 642, 645,
647, 653, 655, 656, 658, 666, 670, 673, 675, 676, 679, 680, 681, 683, 686, 687,
688, 691, 699, 700, 704, 705, 706, 707, 714, 717, 718, 721, 725, 729, 730, 731,
735, 738, 740, 742, 744, 749, 751, 752, 753, 754, 755, 758, 760, 764, 765, 772,
773, 777, 790, 791, 801, 802, 809, 815, 826, 828, 831, 834, 837, 838, 839, 843,
844, 848, 850, 852, 853, 855, 858, 859, 861, 862, 863, 867, 870, 874, 875, 879,
883, 884, 885, 887, 897, 898, 900, 901, 902, 904, 905, 906, 907, 910, 911, 912,
915, 917, 918, 920, 926, 937, 941, 943, 950, 951, 952, 959, 966, 967, 972, 974,
979, 980, 981, 983, 984, 986, 993, 996, 998, 0, 2, 5, 7, 8, 9, 11, 12, 14, 15,
17, 19, 20, 22, 23, 27, 29, 31, 34, 35, 36, 41, 45, 47, 48, 49, 50, 52, 53, 54,
56, 58, 59, 60, 62, 63, 64, 66, 67, 71, 72, 74, 76, 79, 82, 85, 86, 88, 91, 92,
95, 96, 97, 98, 99, 100, 101, 102, 103, 109, 111, 112, 115, 116, 117, 118, 125,
129, 131, 132, 136, 139, 140, 141, 142, 144, 145, 146, 148, 150, 154, 155, 157,
```

```
159, 160, 161, 166, 168, 171, 173, 175, 179, 180, 184, 186, 190, 191, 198, 200,
202, 206, 209, 211, 212, 215, 217, 218, 219, 225, 226, 232, 234, 235, 236, 239,
241, 242, 245, 246, 247, 249, 250, 251, 254, 255, 261, 262, 267, 268, 270, 271,
273, 275, 280, 281, 287, 288, 290, 291, 295, 312, 313, 314, 316, 317, 320, 321,
324, 325, 330, 332, 333, 334, 335, 336, 338, 344, 345, 346, 347, 348, 351, 354,
356, 357, 362, 364, 365, 367, 370, 371, 373, 374, 375, 377, 381, 382, 384, 385,
386, 387, 390, 395, 398, 402, 404, 406, 407, 408, 411, 412, 413, 416, 421, 422,
424, 430, 433, 434, 435, 439, 440, 443, 444, 445, 446, 447, 450, 453, 454, 456,
458, 462, 463, 467, 473, 474, 476, 477, 480, 486, 489, 491, 492, 493, 496, 497,
499, 500, 506, 507, 508, 511, 512, 513, 514, 520, 521, 523, 525, 526, 527, 528,
535, 537, 540, 545, 547, 549, 550, 551, 554, 556, 559, 560, 561, 566, 568, 570,
573, 575, 576, 578, 581, 582, 588, 589, 590, 593, 596, 598, 600, 601, 604, 605,
606, 608, 609, 612, 615, 617, 622, 623, 624, 625, 628, 631, 632, 633, 635, 643,
644, 646, 648, 649, 652, 654, 660, 661, 662, 663, 664, 668, 669, 671, 674, 677,
682, 684, 685, 690, 692, 693, 694, 697, 701, 703, 708, 710, 711, 712, 713, 715,
723, 724, 727, 728, 733, 734, 736, 737, 741, 743, 745, 746, 747, 750, 756, 759,
761, 762, 763, 767, 768, 769, 775, 776, 778, 779, 781, 782, 783, 784, 785, 786,
787, 788, 792, 793, 794, 795, 797, 798, 800, 805, 806, 808, 810, 812, 814, 816,
818, 820, 821, 822, 823, 824, 827, 829, 830, 832, 833, 835, 841, 846, 847, 849,
851, 854, 856, 860, 865, 869, 871, 872, 873, 876, 877, 878, 880, 881, 886, 888,
890, 893, 894, 896, 903, 909, 913, 914, 916, 919, 921, 922, 925, 927, 928, 929,
930, 931, 932, 934, 936, 938, 939, 942, 944, 947, 948, 949, 953, 954, 955, 956,
957, 960, 961, 962, 965, 968, 969, 970, 971, 975, 977, 978, 985, 987, 988, 990,
992, 995, 997, 999]
```

## Task 4: Kernelized SVM

There are 2 hyperparamters to be tuned: C and $\gamma$. These hyperparameters appear in in the Kernelized SVM as follows:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to the constraints:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, n.$$

Here,

- $\alpha_i$ are the Lagrange multipliers
- $y_i \in \{-1, 1\}$ are the class labels
- $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function

- $C$ is the regularization parameter

The Radial Basis Function (RBF) kernel is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

where:

- $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function
- $\mathbf{x}_i, \mathbf{x}_j$ are input feature vectors
- $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ is the squared Euclidean distance
- $\gamma$ is the kernel hyperparameter

These hyperparameters were tuned such that the `train_data` becomes separable and thus the decision boundary is consistent with the `train_data` labels.
These hyperparameters are : C = $10^8$ and $\gamma = 0.01$
Using these hyperparameters, the kernelized SVM was trained on the `train_data` and was tested on the `test_data`. The number of misclassifications was obtained to be 73. This gives rise to a misclassification rate of 36.5%.
The misclassification rate can be attributed to a very high C value which results in very hard margins of the kernelized SVM to the `train_data`

### Task 5: Perceptron on the images which aren't resposnsible for non separability

The misclassified images were obtained from the SVM. These images were removed from the `train_data` and then the perceptron is retrained on the modified `train_data`. The perceptron thus obtained is once again tested on the `test_data`. The convergence of the Perceptron trained on the filtered train data is given by the following graph in Figure 3:
The misclassification rate of the Perceptron is given by the following graph in Figure 4:

## Problem 2:

### Task 1: Multi Layer Perceptron

The input data was split into `train_data` and `test_data` using `train_test_split`. The MLP was trained on the `train_data` and was evaluated on the `test_data`. The metrics obtained on training are given as follows:
**Accuracy:**

$$0.9535$$

**Recall:**

$$\begin{bmatrix} 0.980 & 0.985 & 0.960 & 0.945 & 0.950 & 0.940 & 0.975 & 0.945 & 0.950 & 0.905 \end{bmatrix}$$

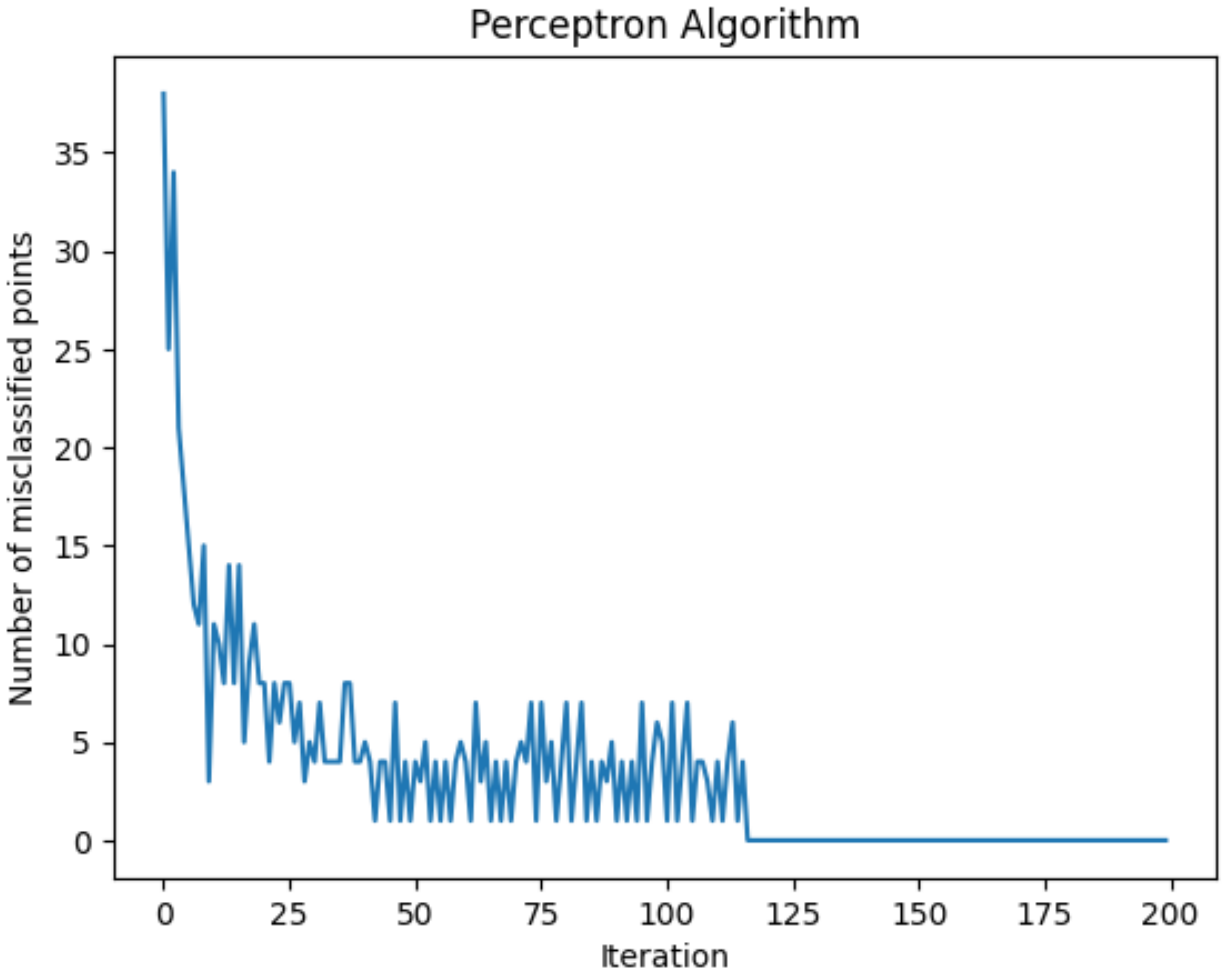


Figure 3: Convergence of perceptron after removal of sources of non separability

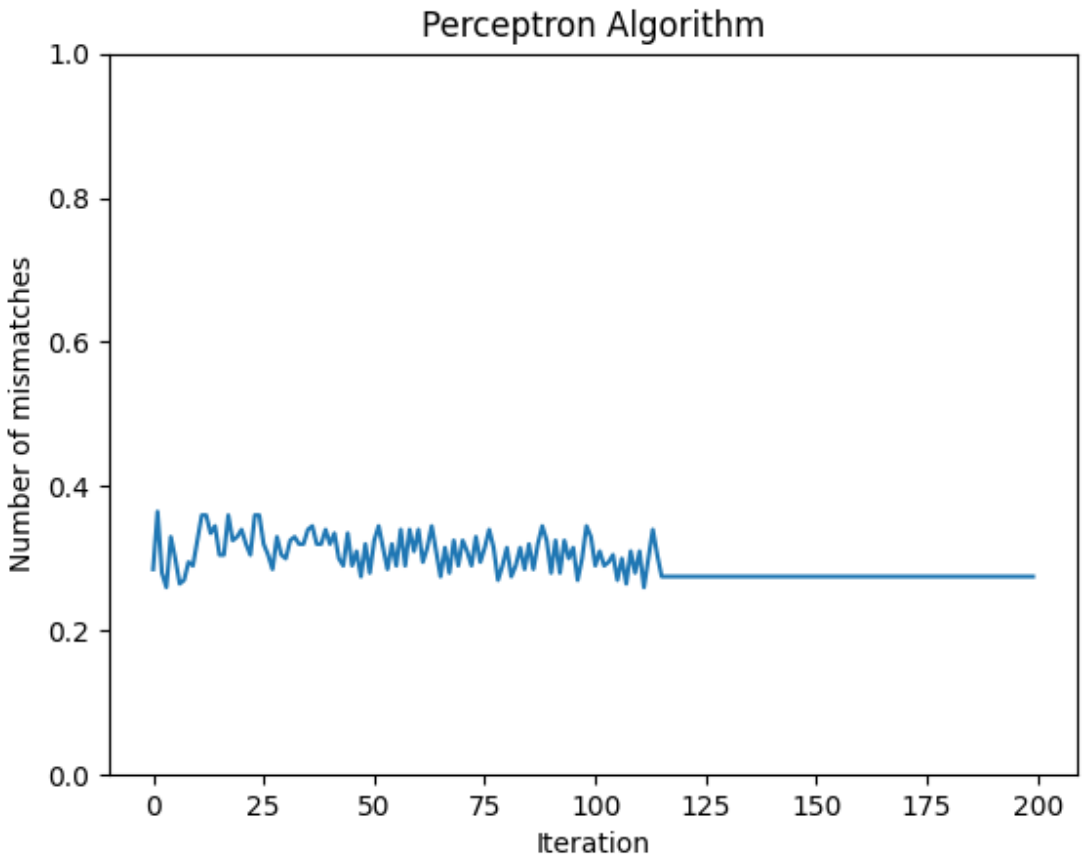


Figure 4: Misclassification rate of perceptron algorithm after sources of non separability removed

**Precision:**

$$\begin{bmatrix}0.9703 & 0.9563 & 0.9366 & 0.9403 & 0.9500 & 0.9592 & 0.9653 & 0.9844 & 0.9406 & 0.9330\end{bmatrix}$$

**F1 Score:**

$$\begin{bmatrix}0.9751 & 0.9704 & 0.9481 & 0.9426 & 0.9500 & 0.9495 & 0.9701 & 0.9643 & 0.9453 & 0.9188\end{bmatrix}$$

**Confusion Matrix:**

$$\begin{bmatrix}196 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0\\ 0 & 197 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1\\ 1 & 0 & 192 & 1 & 0 & 2 & 4 & 0 & 0 & 0\\ 1 & 0 & 5 & 189 & 0 & 1 & 0 & 0 & 2 & 2\\ 0 & 1 & 1 & 1 & 190 & 0 & 2 & 0 & 0 & 5\\ 2 & 1 & 1 & 4 & 1 & 188 & 1 & 1 & 1 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 195 & 0 & 4 & 0\\ 0 & 2 & 2 & 0 & 2 & 0 & 0 & 189 & 0 & 5\\ 2 & 2 & 3 & 0 & 0 & 3 & 0 & 0 & 190 & 0\\ 0 & 3 & 0 & 5 & 6 & 1 & 0 & 2 & 2 & 181\end{bmatrix}$$

## Task 2: CNN

Similar to the MLP, the CNN was trained on the `train_data` and evaluated on the `test_data`. The metrics obtained are given as follows:

**Accuracy:**

$$0.9730$$

**Recall:**

$$\begin{bmatrix}0.980 & 0.990 & 0.985 & 0.975 & 0.975 & 0.975 & 0.975 & 0.990 & 0.980 & 0.905\end{bmatrix}$$

**Precision:**

$$\begin{bmatrix}0.9949 & 0.9851 & 0.9752 & 0.9848 & 0.9420 & 0.9330 & 0.9653 & 0.9851 & 0.9703 & 1.0000\end{bmatrix}$$

**F1 Score:**

$$\begin{bmatrix}0.9874 & 0.9875 & 0.9801 & 0.9799 & 0.9582 & 0.9535 & 0.9701 & 0.9875 & 0.9751 & 0.9501\end{bmatrix}$$

**Confusion Matrix:**

$$\begin{bmatrix}196 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0\\ 0 & 198 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 1 & 197 & 0 & 0 & 0 & 0 & 2 & 0 & 0\\ 0 & 0 & 1 & 195 & 0 & 2 & 0 & 0 & 2 & 0\\ 0 & 2 & 1 & 0 & 195 & 0 & 2 & 0 & 0 & 0\\ 0 & 0 & 0 & 3 & 1 & 195 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 3 & 195 & 0 & 1 & 0\\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 198 & 0 & 0\\ 1 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 196 & 0\\ 0 & 0 & 0 & 0 & 10 & 6 & 0 & 1 & 2 & 181\end{bmatrix}$$

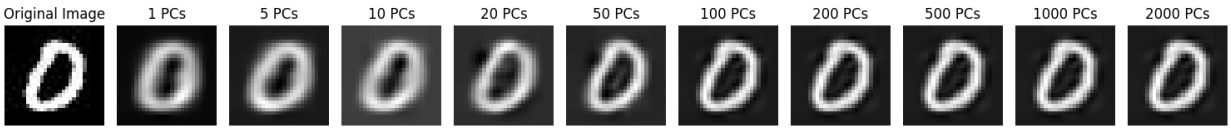


Figure 5: Reconstruction of an image using different number of principal compoenents

## Task 3: PCA

PCA was used to extract the principal components of the data and these principal components were used to transform the data to a lower dimensional space. I have chosen $num\_principal\_components = 100$ as that gave a decent approximation to the original images (by decent I am referring to the task of reconstructing the image from the trasnformed data. $num\_principal\_components = 100$ restored the image decently well).
An image was selected randomly and was restored from its projected version. The variation in the restoration using diferent number of principal components is given below in Figure 5:

## Task 4: MLP with PCA features

The MLP was again trained this time using the transformed data as the input. The metrics as obtained on the test data are given below:

**Accuracy**

$$0.9524999856948853$$

**Recall**

$$\begin{bmatrix}0.9712 & 0.9703 & 0.9583 & 0.9303 & 0.9617 & 0.9086 & 0.9858 & 0.9363 & 0.9387 & 0.9600\end{bmatrix}$$

**Precision**

$$\begin{bmatrix}0.9806 & 0.9899 & 0.9340 & 0.9791 & 0.9526 & 0.9389 & 0.9541 & 0.9646 & 0.9299 & 0.8984\end{bmatrix}$$

**F1 Score**

$$\begin{bmatrix}0.9758 & 0.9800 & 0.9460 & 0.9541 & 0.9571 & 0.9235 & 0.9697 & 0.9502 & 0.9343 & 0.9282\end{bmatrix}$$

**Confusion Matrix**

$$\begin{bmatrix} 202 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 3 & 0 \\ 0 & 196 & 1 & 2 & 0 & 0 & 1 & 0 & 2 & 0 \\ 1 & 0 & 184 & 1 & 3 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 5 & 187 & 0 & 4 & 0 & 0 & 4 & 1 \\ 0 & 0 & 1 & 0 & 201 & 0 & 2 & 0 & 0 & 5 \\ 1 & 0 & 0 & 0 & 2 & 169 & 4 & 2 & 5 & 3 \\ 1 & 0 & 0 & 0 & 1 & 1 & 208 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 & 2 & 1 & 0 & 191 & 0 & 5 \\ 0 & 0 & 1 & 1 & 0 & 4 & 1 & 1 & 199 & 5 \\ 1 & 0 & 0 & 0 & 2 & 1 & 0 & 2 & 1 & 168 \end{bmatrix}$$

## Task 5: Logistic regression using PCA features

We now train a logistic regression model on the PCA features using 2 different approaches:

- Multi class classification
- One vs rest

For the logistic regression model trained for multi class classification, here are the performance metrics:

**Accuracy**

$$0.8875$$

**Recall**

$$\begin{bmatrix} 0.9747 & 0.9492 & 0.8698 & 0.9110 & 0.8798 & 0.7818 & 0.9677 & 0.8952 & 0.7766 & 0.8718 \end{bmatrix}$$

**Precision**

$$\begin{bmatrix} 0.9019 & 0.8738 & 0.9330 & 0.8832 & 0.8656 & 0.8866 & 0.9052 & 0.9082 & 0.8743 & 0.8416 \end{bmatrix}$$

**F1 Score**

$$\begin{bmatrix} 0.9369 & 0.9100 & 0.9003 & 0.8969 & 0.8726 & 0.8309 & 0.9354 & 0.9017 & 0.8226 & 0.8564 \end{bmatrix}$$

**Confusion Matrix**

$$\begin{bmatrix} 193 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 0 \\ 1 & 187 & 2 & 1 & 0 & 2 & 0 & 0 & 4 & 0 \\ 1 & 3 & 167 & 3 & 1 & 0 & 4 & 6 & 4 & 3 \\ 0 & 1 & 3 & 174 & 0 & 5 & 2 & 2 & 2 & 2 \\ 0 & 6 & 2 & 0 & 161 & 0 & 1 & 0 & 1 & 12 \\ 9 & 3 & 0 & 9 & 4 & 172 & 11 & 2 & 7 & 3 \\ 2 & 1 & 1 & 0 & 1 & 1 & 210 & 0 & 1 & 0 \\ 2 & 0 & 3 & 0 & 7 & 0 & 0 & 188 & 0 & 10 \\ 3 & 12 & 1 & 6 & 4 & 11 & 2 & 3 & 153 & 2 \\ 3 & 1 & 0 & 4 & 8 & 2 & 0 & 6 & 1 & 170 \end{bmatrix}$$

We have also trained 10 binary classifiers using the one vs rest approach. The average AUC scores and the ROC graphs are attached below in Figure 6:

# Problem 3:

## 3.1 : Linear Regression

### Task 1: X does not have full column rank

No we can't do OLS is X doesn't have full column rank as $X^TX$ becomes singular( proof below). To do OLS we can do regularisation so that $X^TX$ becomes invertible. Since $X$ does not have full column rank, there exists a nonzero vector $v \in \mathbb{R}^n$ such that:

$$Xv = 0.$$

Taking the transpose on both sides,

$$(Xv)^T = 0^T.$$

Using the property $(AB)^T = B^TA^T$, we obtain:

$$v^TX^T = 0^T.$$

Multiplying both sides by $X$, we get:

$$v^TX^TX = 0^T.$$

Since $v \neq 0$, this implies that $X^TX$ has a nonzero vector in its null space, meaning:

$$X^TXv = 0.$$

By the definition of singularity, a square matrix is singular if and only if it has a nontrivial null space. Since we have found such a $v \neq 0$, it follows that $X^TX$ is singular.

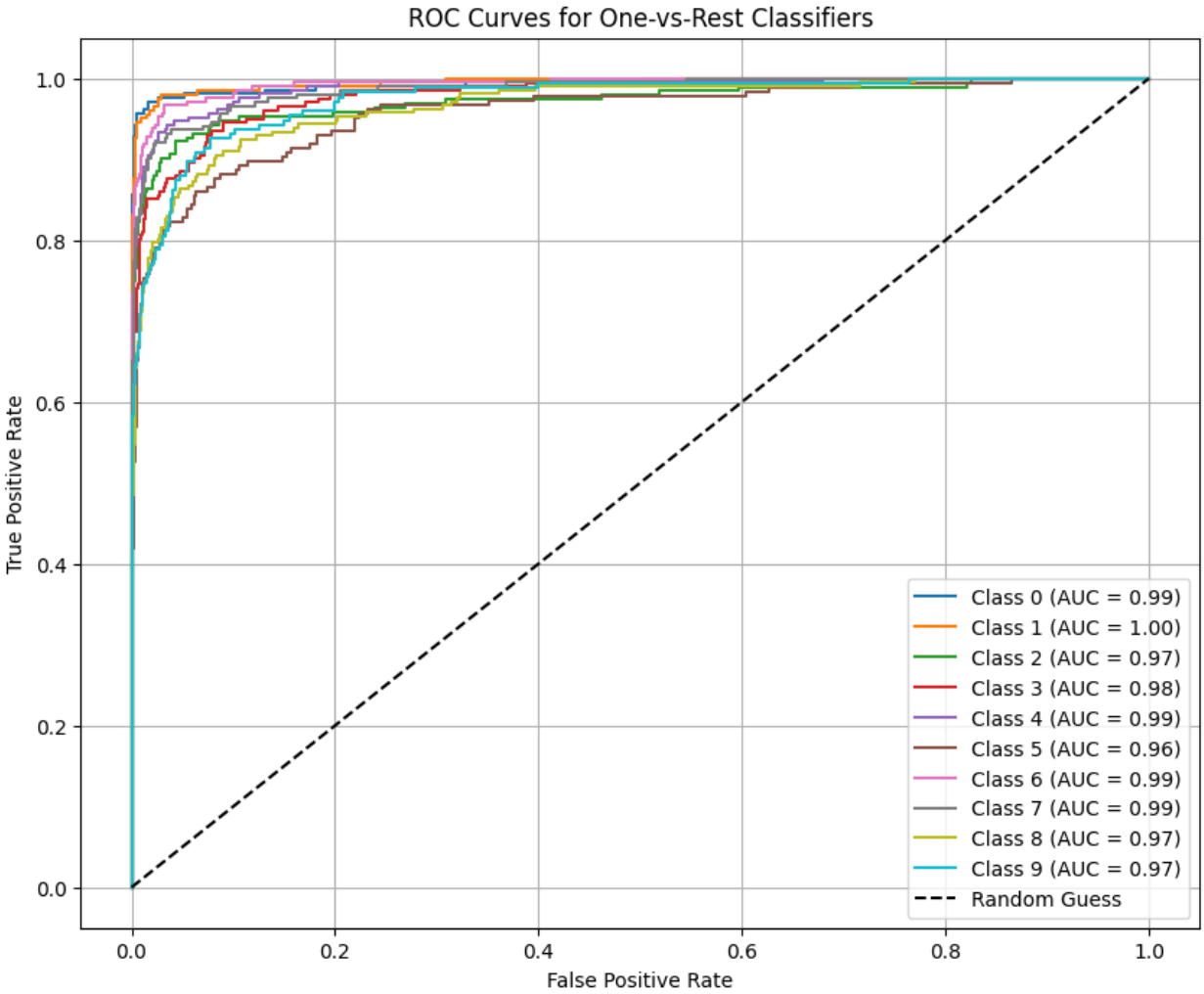


Figure 6: Average AUC and the ROC graphs for all 10 binary classifiers

**Task 2: MSE on $\mathcal{D}_1^{train}$**

Using OLS we get the following performance metrics:
**MSE** = 0.0437017962349622

$$w_1^{ols} = \begin{bmatrix} 0.3217 \\ 0.2197 \\ 0.0891 \\ 0.1084 \\ 0.4806 \end{bmatrix}$$

Using ridge regression we get the following performance metrics
**MSE** = 0.05057778676155785

$$w_1^{rr} = \begin{bmatrix} 0.3189 \\ 0.2133 \\ 0.0808 \\ 0.1025 \\ 0.4572 \end{bmatrix}$$

**Task 3: MSE on $\mathcal{D}_2^{train}$**

Using OLS we get **MSE** = 282396.26390835823
Using Ridge regression we get **MSE** = 10.352926835603585
$w_2^{rr}$ and $w_2^{ols}$ have been attached as csv files in the format prescribed in the assignment.

## 3.2: Support Vector Regression

I received the stock 'MSFT' from the oracle. The graphs on the test set for the linear SVR problem using the 3 window sizes are attached below in Figures 7-21. Note that in all the figures the stock prices are plotted in the original scale and not the normalised values.

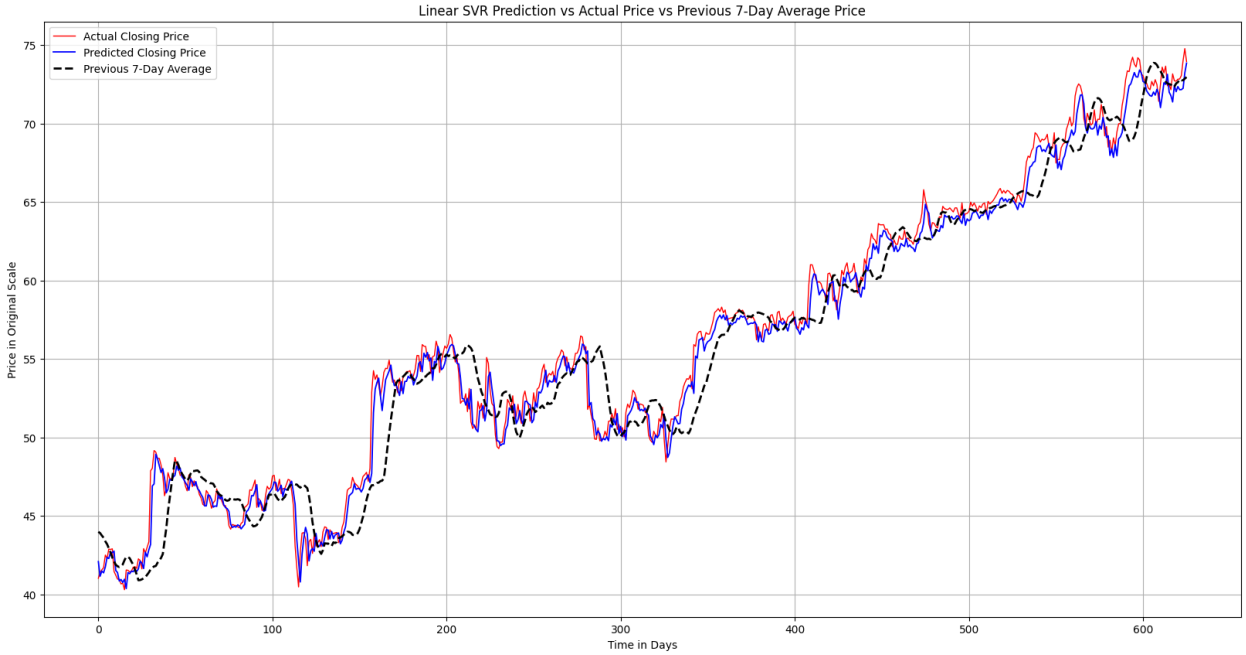


Figure 7: Linear SVR using window size 7

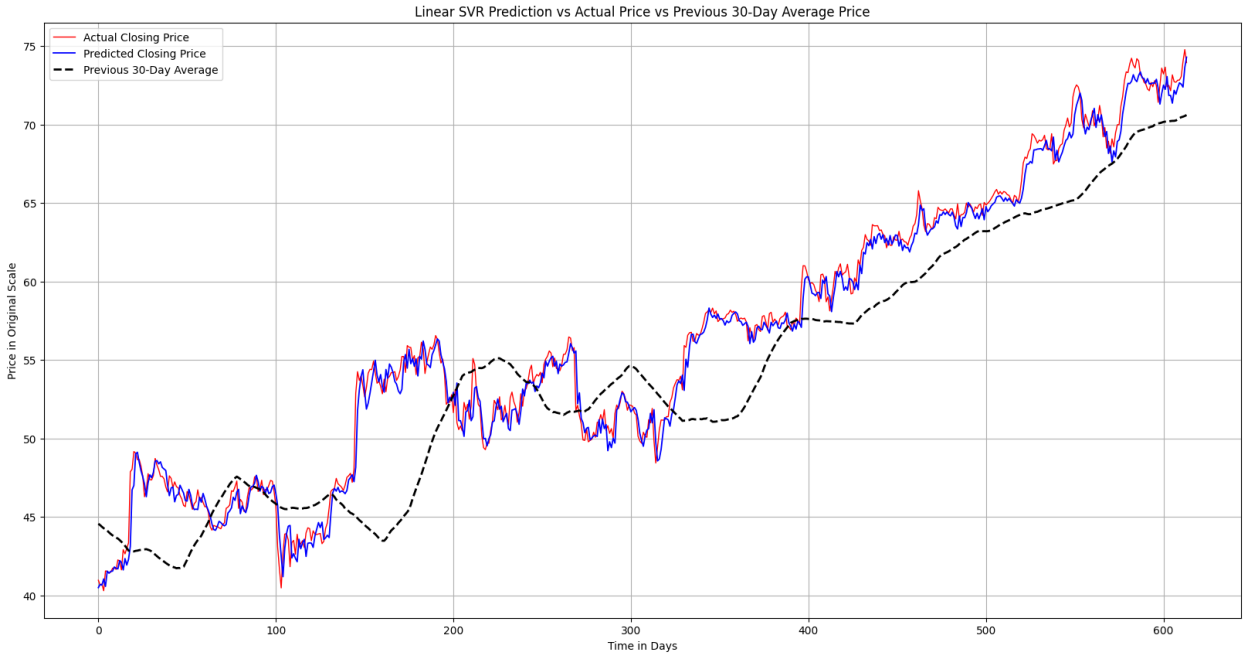


Figure 8: Linear SVR using window size 30

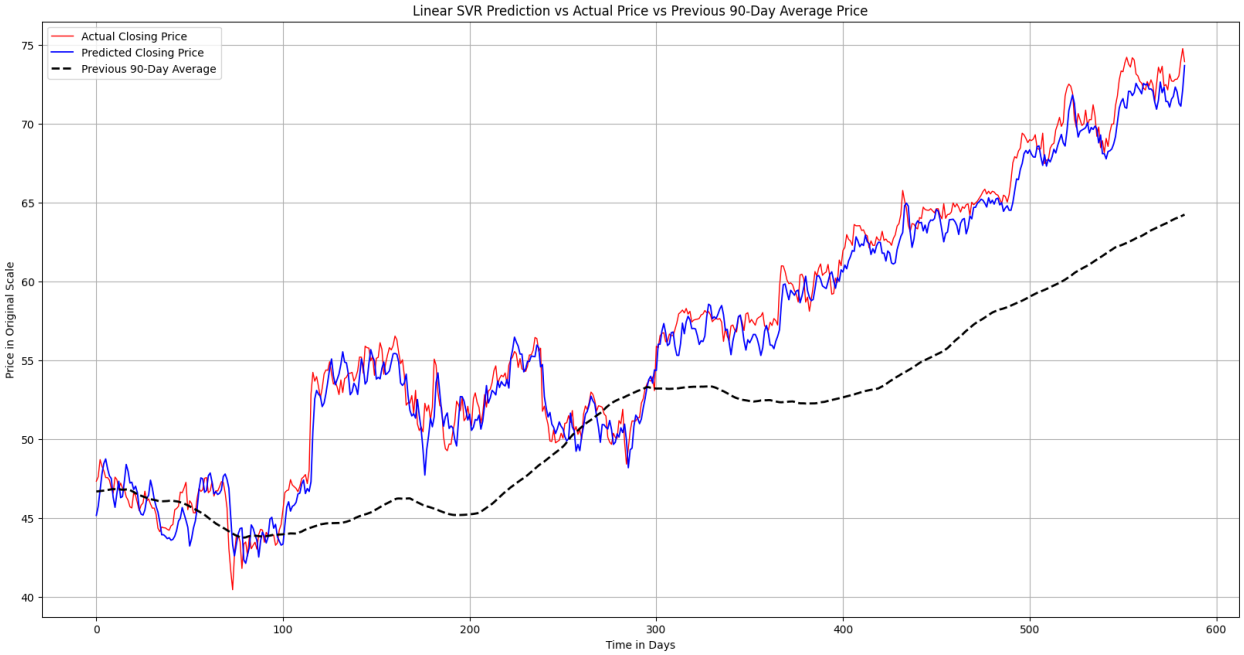


Figure 9: Linear SVR using window size 90

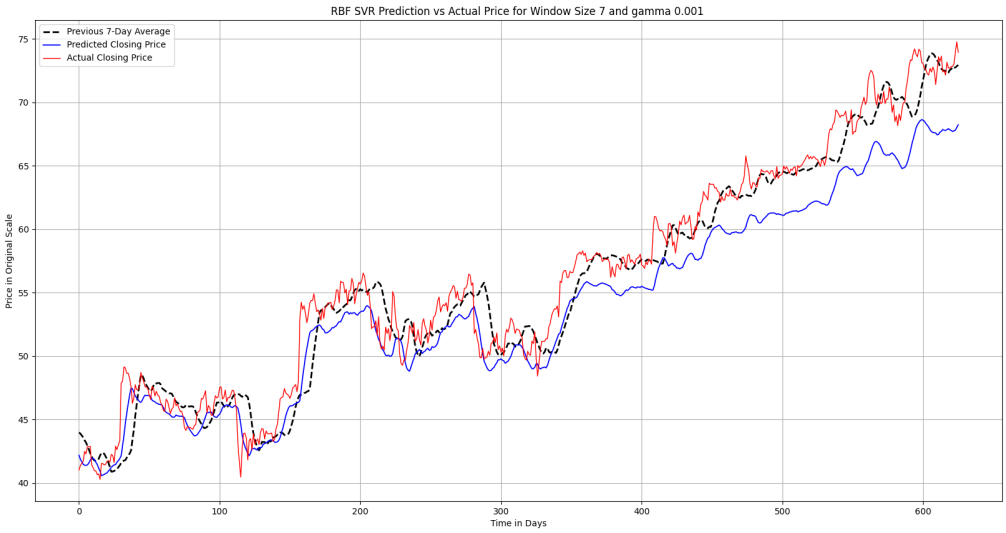


Figure 10: Kernel SVR using window size 7 and gamma 0.001

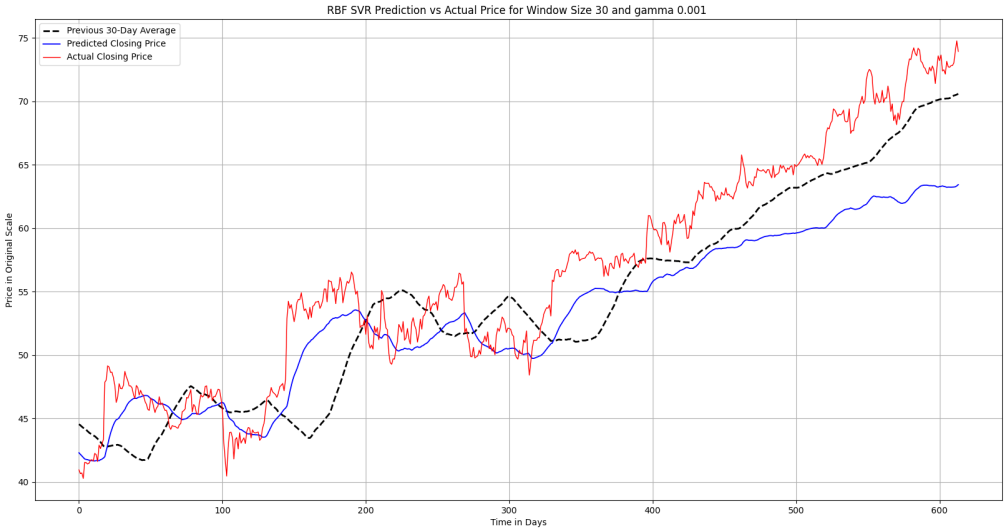


Figure 11: Kernel SVR using window size 30 and gamma 0.001

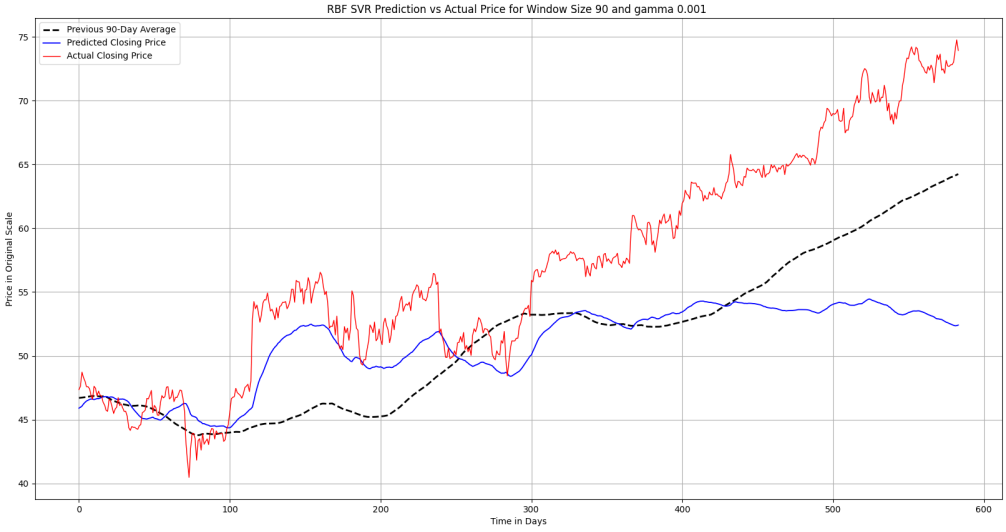


Figure 12: Kernel SVR using window size 90 and gamma 0.001

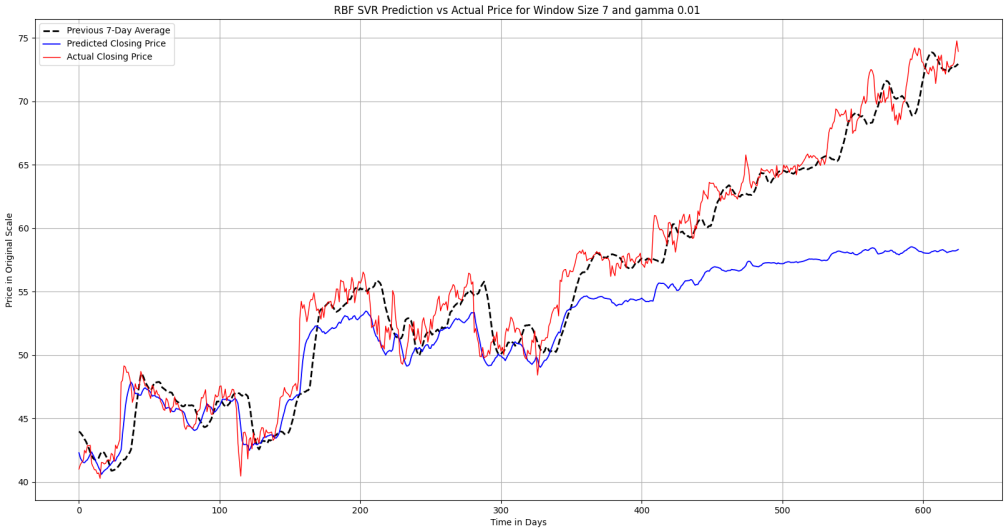


Figure 13: Kernel SVR using window size 7 and gamma 0.01

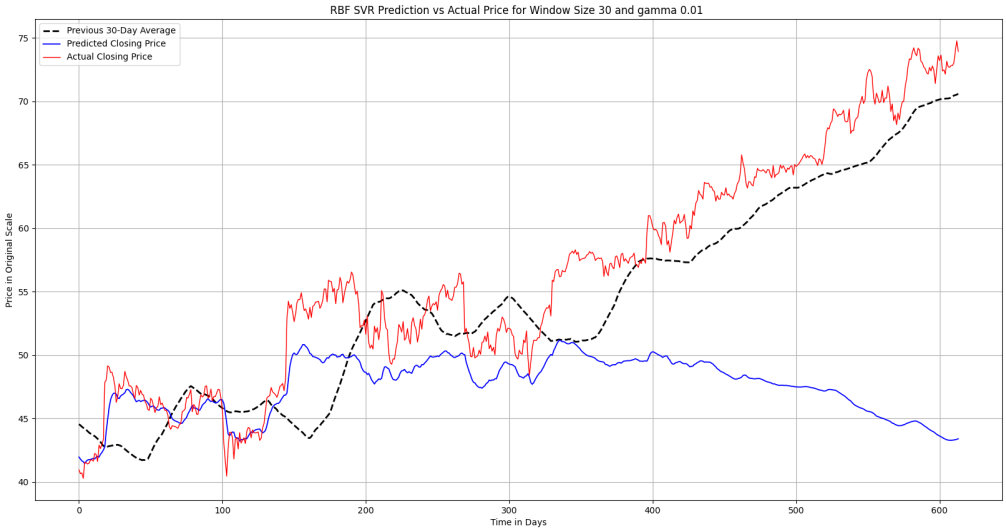


Figure 14: Kernel SVR using window size 30 and gamma 0.01

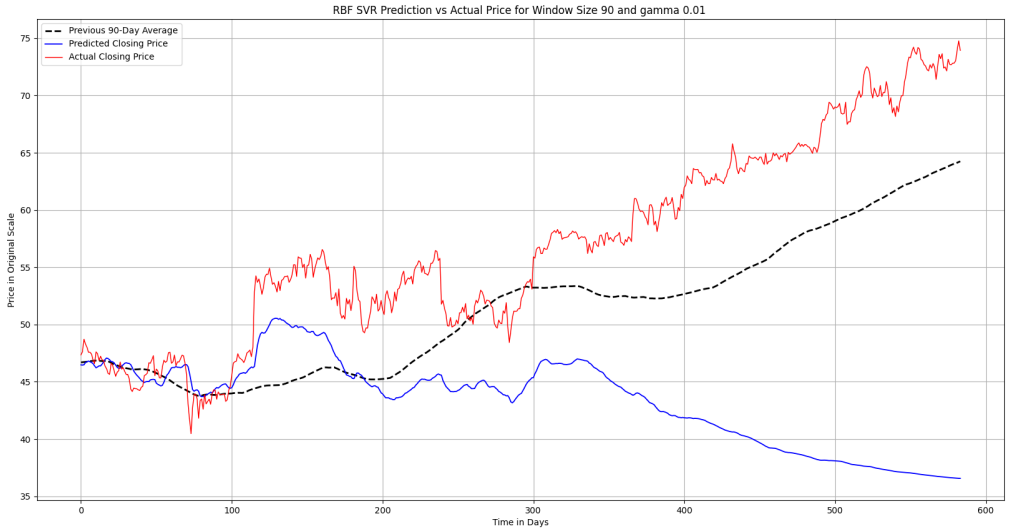


Figure 15: Kernel SVR using window size 90 and gamma 0.01

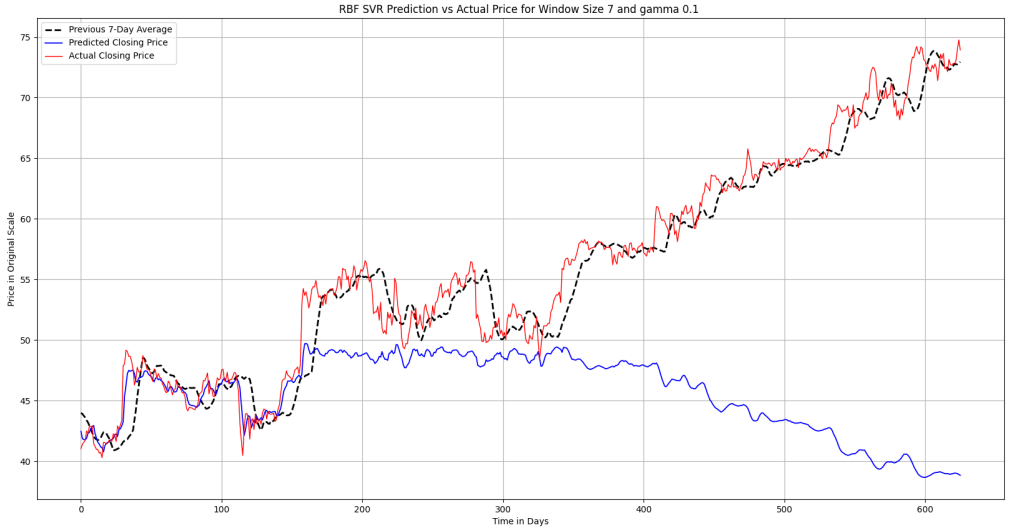


Figure 16: Kernel SVR using window size 7 and gamma 0.1

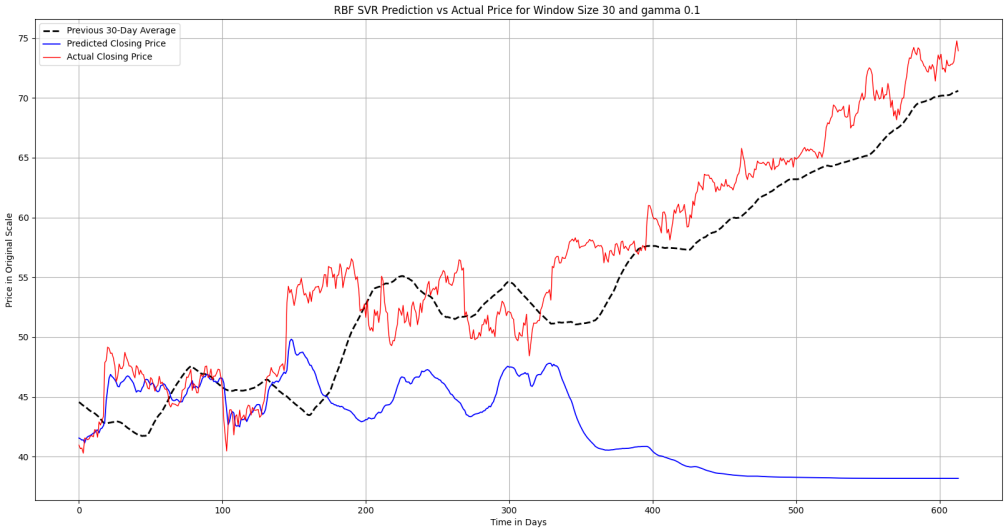


Figure 17: Kernel SVR using window size 30 and gamma 0.1

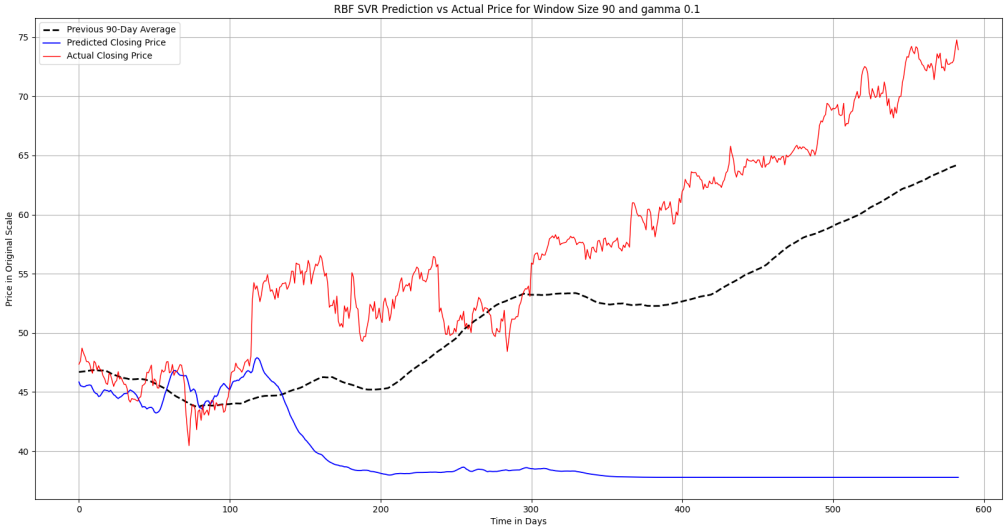


Figure 18: Kernel SVR using window size 90 and gamma 0.1

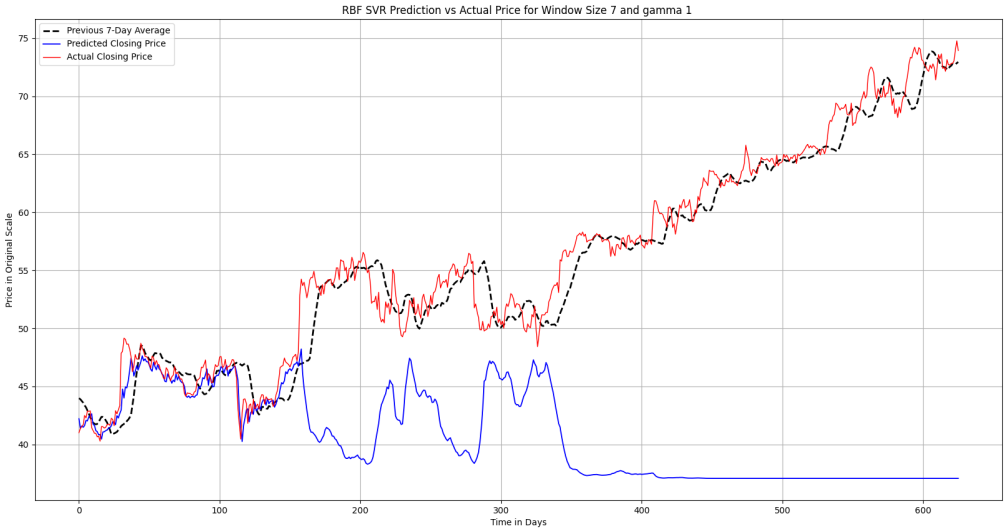


Figure 19: Kernel SVR using window size 7 and gamma 1

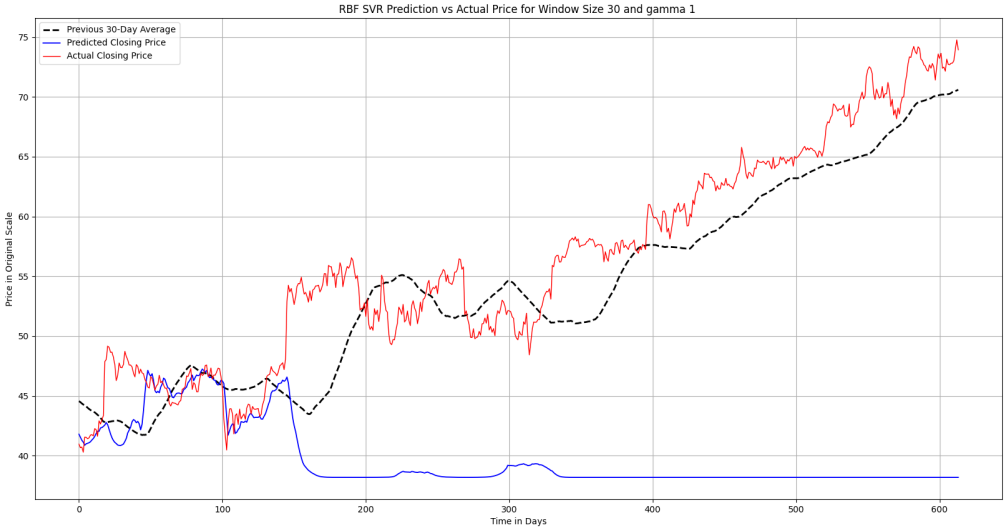


Figure 20: Kernel SVR using window size 30 and gamma 1

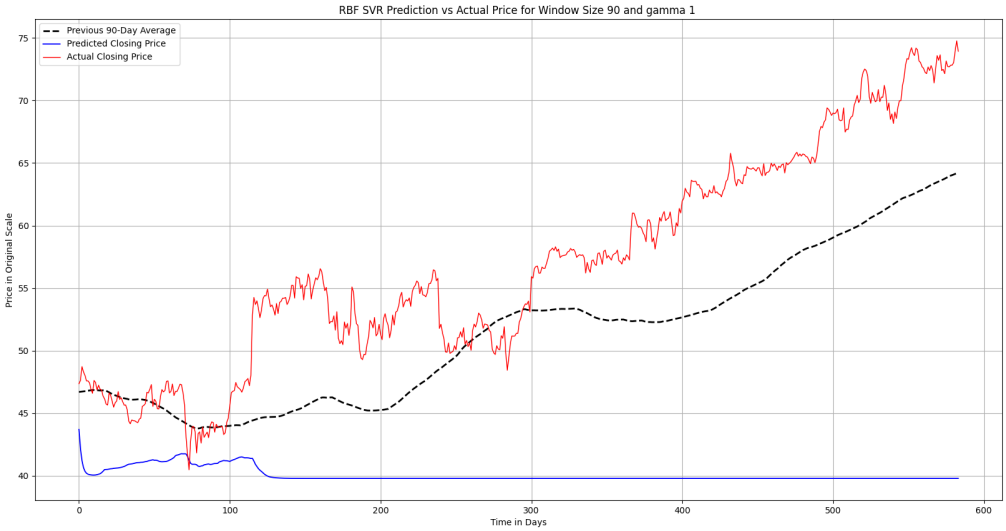


Figure 21: Kernel SVR using window size 90 and gamma 1