Artificial Intelligence and Machine Learning

Assignment 02

Instructor : Prof. Chiranjib Bhattacharyya

Student : Kuldeep Jatav

SR Number: 23684

INDIAN INSTITUTE OF SCIENCE

भारतीय विज्ञान संस्थान

Solution 1 : Support Vector Machine and Perceptron

- **Tasks**

  1. Perceptron algorithm does not seem converging on the given dataset maybe the dataset is not linearly separable, to be more sure i ran the perceptron algorithm for 100000 iterations still the lowest misclassification rate achieved was around 0.193, and in the next part when we remove the points which are causing non-separability the algorithm converged in ≈ 300 iterations

  2. Slack support vector machine with linear kernel

  – Primal version

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

subject to

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall i = 1,\dots,n$$
$$\xi_i \geq 0, \quad \forall i = 1,\dots,n$$

  – Dual version

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

subject to

$$0 \leq \alpha_i \leq C, \quad \forall i = 1,\dots,n$$

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

  while solving both problems i used **cvxopt** for convex optimization with $C = 1$

  3. In this part i used Gaussian kernel with $\gamma = 0.1$ and $C = 1.0$

$$K(x,z) = exp\left(-\frac{||x-z||^2}{2\sigma^2}\right) \quad , \quad \gamma = \frac{1}{2\sigma^2}$$

  4. In this part first i make *new_data* by removing the points which were causing non-separability and then again ran the perceptron algorithm on this new data it converged in ≈ 300 iterations

- **Deliverables**

  1. Plot of misclassification rate vs number of iterations for perceptron algorithm on given dataset

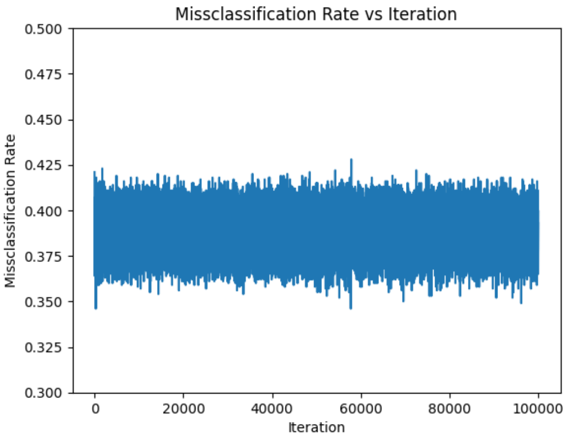


Figure 1: $\mathbf{10^5}$ **Iterations**

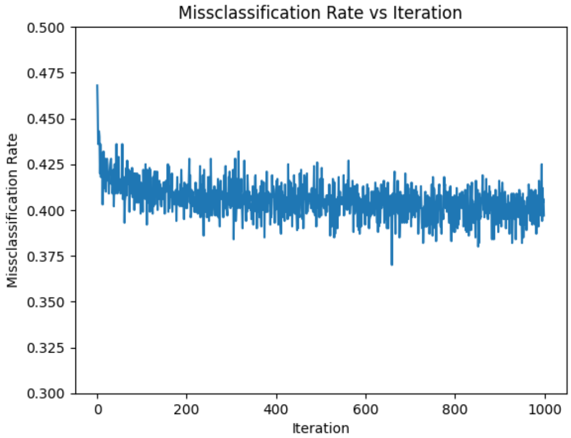


Figure 2: $\mathbf{10^3}$ **Iterations**

Figure 3: Two adjacent images

2. For the given dataset so dual version is faster than primal version because

- Primal time : ≈ 1.24 `seconds`
- Dual time : ≈ 1.17 `seconds`

3. For images that cause non-separability refer `inseparable_23684.csv` (attached in submission).

4. Final misclassification rate for the kernelized SVM

- 145 out of 1000 train points were misclassified
- 19 out of 200 test points were misclassified

5. plot of misclassification rate vs number of iterations for perceptron algorithm on given dataset after removing the points which were causing non-separability

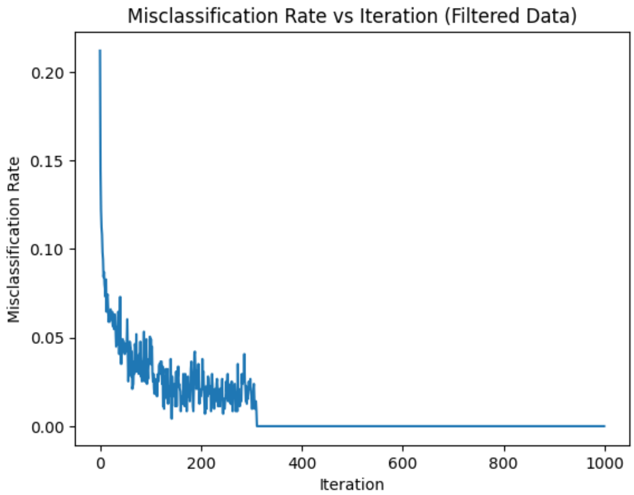


Figure 4: Misclassification rate vs number of iterations

Solution 2 : Logistic Regression, MLP, CNN & PCA

- **Tasks**

1. In this part we constructed an MLP model with following architecture

- Input layer : 784 neurons
- Hidden layer 1 : 512 neurons with ReLU activation
- Hidden layer 2 : 256 neurons with ReLU activation
- Output layer : 10 neurons with softmax activation
- Loss function : Cross entropy loss

On running the code for 20 epochs with **80:20** split then i got $\approx 94\%$ accuracy on test data

2. In this part i constructed a CNN model that takes 28×28 as input & outputs 10 class probabilities.

- Layer 1 (CNN) : 32 filters of size 3×3 with ReLU activation
- Layer 2 (CNN) : 64 filters of size 3×3 with ReLU activation
- Layer 3 (Max_Pooling) : 2×2 with stride 2
- Layer 4 (Fully Connected) : 128 neurons with ReLU activation
- Layer 5 (Fully Connected) : 10 neurons with softmax activation
- Loss function : Cross entropy loss

On running the code for 20 epochs with **80:20** split then i got $\approx 97\%$ accuracy on test data

3. Image feature extraction using PCA here is the plot of first 8 PCA and refer `AIML_2025_A2_23684.py`

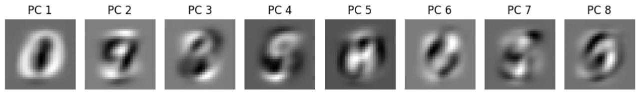


Figure 5: 8-PCA plot

4. In this part i again trained same multilayer perceptron as in part 1 but used 50 PCA features as input and got ccuracy on test data : $\approx 95\%$, for code refer `AIML_2025_A2_23684.py`

5. In this part of the question i trained a logistic regression model for multiclass classification abd then train a model binary classifier `one_vs_rest()`

- **Deliverables**

1. I reconstructed image of digit 8 using different number of principal components

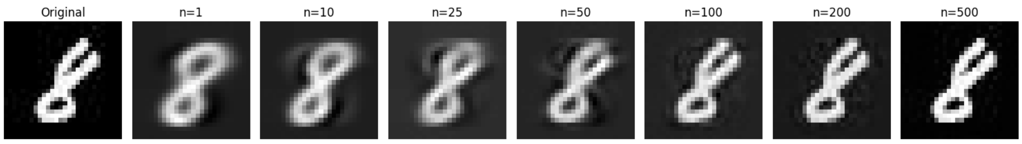


Figure 6: Reconstructed image of digit 8

from the plot we can see that as we increase the number of principal components the image is getting clearer and clearer. specifically when we use 100+ the image is almost identical to the original image.

3. Here is the ROC plot for each class for `one_vs_rest()` classifier also average AUC i got is 0.88

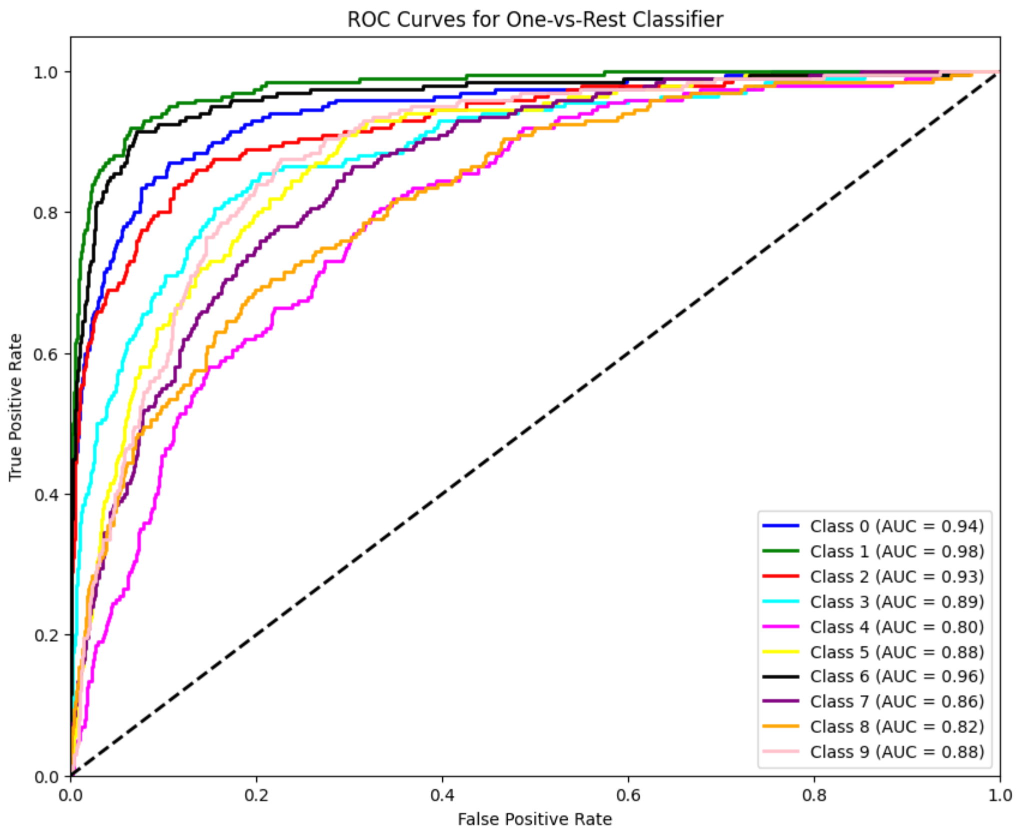


Figure 7: ROC plot for each class

2. Metrics for each model for each class as well are as follows

```
MLP Confusion Matrix:
[[200   0   0   0   0   0   0   0   0   0]
 [  0 198   0   0   0   0   1   0   0   1]
 [  1   0 195   0   0   0   1   0   3   0]
 [  1   1   2 191   0   2   2   0   1   0]
 [  0   0   0   0 196   0   1   0   0   3]
 [  0   0   0   0   0 199   0   0   1   0]
 [  0   0   0   0   0   0 200   0   0   0]
 [  3   2   2   1   1   0   0 191   0   0]
 [  0   0   0   1   0   3   0   1 195   0]
 [  4   0   1   2   8   0   2   1   2 180]]


Class 0: Precision=0.9569, Recall=1.0000, F1=0.9780
Class 1: Precision=0.9851, Recall=0.9900, F1=0.9875
Class 2: Precision=0.9750, Recall=0.9750, F1=0.9750
Class 3: Precision=0.9795, Recall=0.9550, F1=0.9671
Class 4: Precision=0.9561, Recall=0.9800, F1=0.9679
Class 5: Precision=0.9755, Recall=0.9950, F1=0.9851
Class 6: Precision=0.9662, Recall=1.0000, F1=0.9828
Class 7: Precision=0.9896, Recall=0.9550, F1=0.9720
Class 8: Precision=0.9653, Recall=0.9750, F1=0.9701
Class 9: Precision=0.9783, Recall=0.9000, F1=0.9375

MLP Overall Averages:
Precision: 0.9728
Recall: 0.9725
F1: 0.9723
```

```
CNN Confusion Matrix:
[[200   0   0   0   0   0   0   0   0   0]
 [  0 198   0   0   1   0   0   1   0   0]
 [  0   0 200   0   0   0   0   0   0   0]
 [  0   0   0 197   0   3   0   0   0   0]
 [  0   0   0   0 199   0   0   0   0   1]
 [  0   0   0   0   0 200   0   0   0   0]
 [  0   0   0   0   1   0 199   0   0   0]
 [  0   0   0   0   0   0   0 200   0   0]
 [  0   0   0   1   0   2   1   0 195   1]
 [  0   0   0   0   0   1   0   0   0 199]]


Class 0: Precision=1.0000, Recall=1.0000, F1=1.0000
Class 1: Precision=1.0000, Recall=0.9900, F1=0.9950
Class 2: Precision=1.0000, Recall=1.0000, F1=1.0000
Class 3: Precision=0.9949, Recall=0.9850, F1=0.9899
Class 4: Precision=0.9900, Recall=0.9950, F1=0.9925
Class 5: Precision=0.9709, Recall=1.0000, F1=0.9852
Class 6: Precision=0.9950, Recall=0.9950, F1=0.9950
Class 7: Precision=0.9950, Recall=1.0000, F1=0.9975
Class 8: Precision=1.0000, Recall=0.9750, F1=0.9873
Class 9: Precision=0.9900, Recall=0.9950, F1=0.9925

CNN Overall Averages:
Precision: 0.9936
Recall: 0.9935
F1: 0.9935
```

```
MLP+PCA Confusion Matrix:
[[193   0   0   0   0   0   7   0   0   0]
 [  0 193   1   1   1   0   0   1   1   2]
 [  1   1 188   3   0   0   2   2   3   0]
 [  1   1   4 187   0   1   2   3   0   1]
 [  0   1   2   2 180   0   3   0   0  12]
 [  0   2   0   3   0 186   3   1   3   2]
 [  0   0   0   1   0   2 197   0   0   0]
 [  0   2   0   1   0   0   0 190   0   7]
 [  1   1   3  10   1   7   1   0 176   0]
 [  0   0   1   1  16   0   2   2   0 178]]


Class 0: Precision=0.9847, Recall=0.9650, F1=0.9747
Class 1: Precision=0.9602, Recall=0.9650, F1=0.9626
Class 2: Precision=0.9447, Recall=0.9400, F1=0.9424
Class 3: Precision=0.8947, Recall=0.9350, F1=0.9144
Class 4: Precision=0.9091, Recall=0.9000, F1=0.9045
Class 5: Precision=0.9490, Recall=0.9300, F1=0.9394
Class 6: Precision=0.9078, Recall=0.9850, F1=0.9448
Class 7: Precision=0.9548, Recall=0.9500, F1=0.9524
Class 8: Precision=0.9617, Recall=0.8800, F1=0.9191
Class 9: Precision=0.8812, Recall=0.8900, F1=0.8856

MLP+PCA Overall Averages:
Precision: 0.9348
Recall: 0.9340
F1: 0.9340
```

```
Logistic Regression+PCA Confusion Matrix:
[[185   0   4   2   1   8   0   0   0   0]
 [  0 177   6   0   3   0   0   5   8   1]
 [  5   3 157   3   1   3  13   4  10   1]
 [  4   1  11 153   0  14   3   3   7   4]
 [  1   1   3   5 147   2   9   4   2  26]
 [  5   5   2  10   6 153   0   3  13   3]
 [  1   0   2   0   0   6 190   1   0   0]
 [  0   0  11   4   1   0   0 174   0  10]
 [  6  10   4   4   3  13   3   0 151   6]
 [  2   1   1   1  14   4   1  11   2 163]]


Class 0: Precision=0.8852, Recall=0.9250, F1=0.9046
Class 1: Precision=0.8939, Recall=0.8850, F1=0.8894
Class 2: Precision=0.7811, Recall=0.7850, F1=0.7830
Class 3: Precision=0.8407, Recall=0.7650, F1=0.8010
Class 4: Precision=0.8352, Recall=0.7350, F1=0.7819
Class 5: Precision=0.7537, Recall=0.7650, F1=0.7593
Class 6: Precision=0.8676, Recall=0.9500, F1=0.9069
Class 7: Precision=0.8488, Recall=0.8700, F1=0.8593
Class 8: Precision=0.7824, Recall=0.7550, F1=0.7684
Class 9: Precision=0.7617, Recall=0.8150, F1=0.7874

Logistic Regression+PCA Overall Averages:
Precision: 0.8250
Recall: 0.8250
F1: 0.8241
```

- Top Left image is for MLP model
- Top Right image is for CNN model
- Bottom Left image is for PCA + MLP model
- Bottom Right image is for Logistic Regression model

Solution 3 : Regression

○ **Linear Regression**

- **Tasks**
  1. Query for data done in code! refer `AIML_2025_A2_23684.py`
  2. Solved the linear regression and ridge regression problem using the given dataset $\mathcal{D}_1$ and $\mathcal{D}_2$ for $\lambda = 1$
     - Ordinary least square regression

$$\boxed{J(w) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - w^T x_i)^2 = \frac{1}{2n}\|Y - Xw\|^2}$$

where $X \in \mathbb{R}^{n\times d}$, $Y \in \mathbb{R}^n$, $x^{(i)}, \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$

     - Ridge regression

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

where $X \in \mathbb{R}^{n\times d}$, $Y \in \mathbb{R}^n$, $x^{(i)}, \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$

     while solving both problems i used **cvxopt** for convex optimization with $C = 1$
  3. MSE for w_1^{ols}, w_1^{rr} and w_2^{ols}, w_2^{rr} for dataset $\mathcal{D}_1$ and $\mathcal{D}_2$ are as follows
     - For dataset $\mathcal{D}_1$
       * Ordinary least square regression method : 0.058230037381664414
       * Ridge regression method : 0.05123724209499151

     For dataset $\mathcal{D}_2$
       * Ordinary least square regression method : 65556.6898497358
       * Ridge regression method : 10.748440456477965
- **Deliverables**
  1. Final solution that we get for ordinary least square regression method is

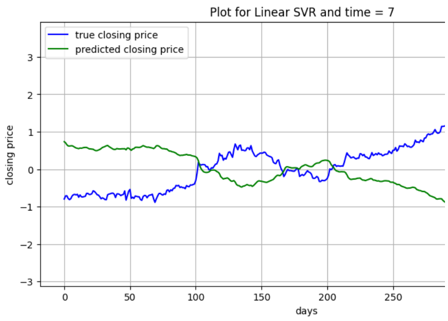
$$w = (X^T X)^{-1} X^T Y$$

  but if X is not full rank then $(X^T X)^{-1}$ will not exist we can solve this problem by using ridge regression method or stochastic gradient descent method.

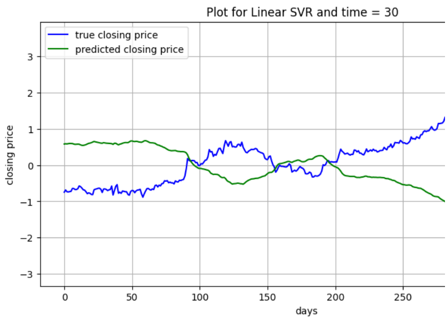
  2. MSE results are mentioned above in task 3 and w_1^{ols}, w_1^{rr} are below
     - $w_1^{ols} = [[-0.24396582], [-1.37368806], [0.192258012], [0.10943952], [-0.05493344], [0.18603667], [0.69107121], [0.2482359], [0.08276927], [-0.0738674]]$
     - $w_1^{rr} = [[-0.25445716], [-1.3052957], [0.28605436], [0.07646213], [-0.09351605], [0.15510994], [0.63710072], [0.22491536], [0.06385936], [-0.08760024]]$
  3. MSE results are mentioned above in task 3 and w_2^{ols}, w_2^{rr} are in `w_ols_23684.csv` , `w_rr_23684.csv` respectively which are attached in submission on teams.

○ **Support Vector Regression**

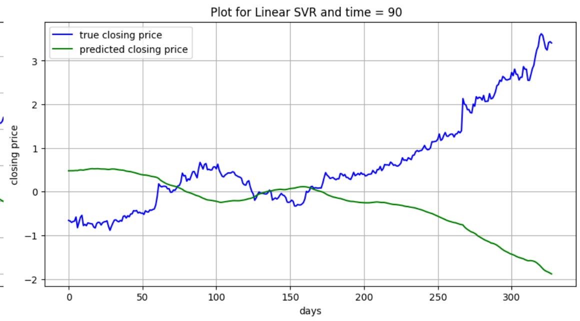
- **Tasks**
  1. Solved the dual slack support vector regression using **cvxopt** refer `AIML_2025_A2_23684.py`
  2. Solved the dual kernelized support vector regression using **cvxopt** refer `AIML_2025_A2_23684.py`
- **Deliverables :** Plot of predicted values , average values and actual values for each γ and *time* are below



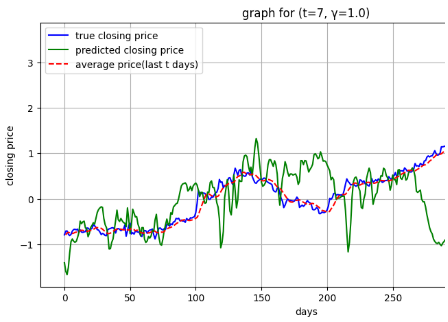
Linear SVR (t=7)



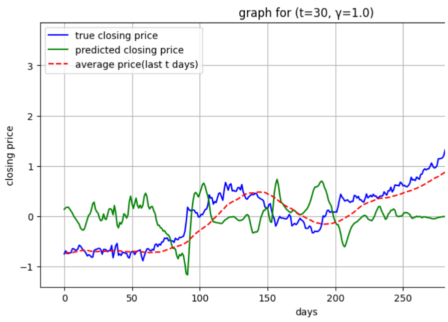
Linear SVR (t=30)



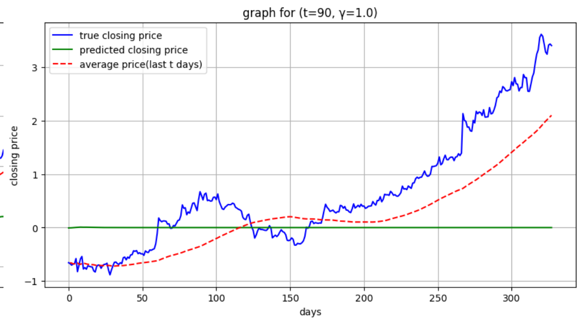
Linear SVR (t=90)



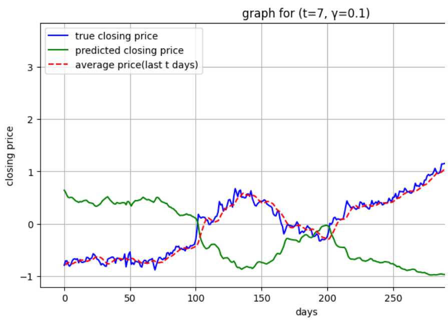
Kernel SVR (t=7, $\gamma = 1$)



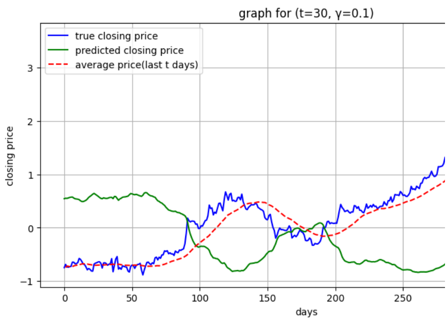
Kernel SVR (t=30, $\gamma = 1$)



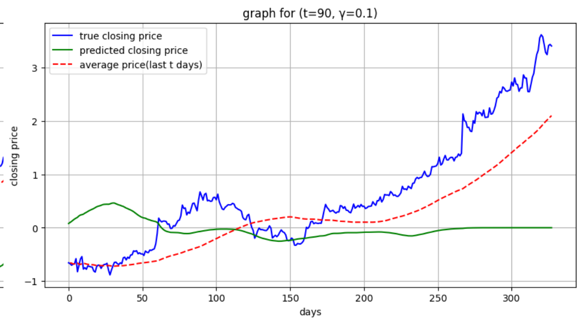
Kernel SVR (t=90, $\gamma = 1$)



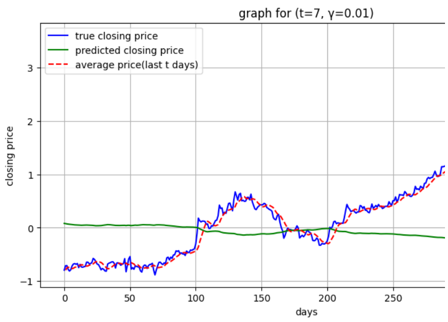
Kernel SVR (t=7, $\gamma = 0.1$)



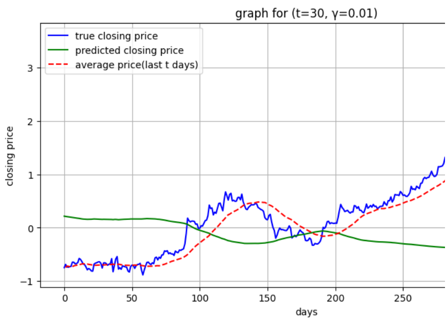
Kernel SVR (t=30, $\gamma = 0.1$)



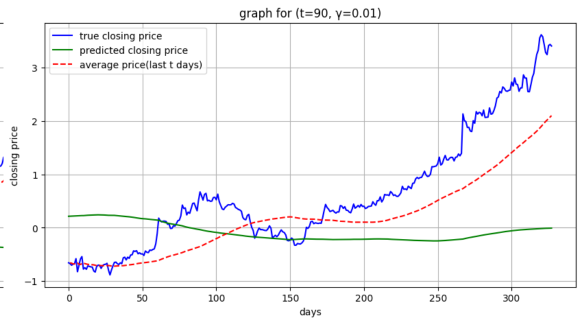
Kernel SVR (t=90, $\gamma = 0.1$)



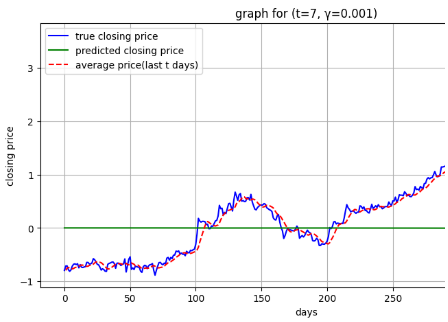
Kernel SVR (t=7, $\gamma = 0.01$)



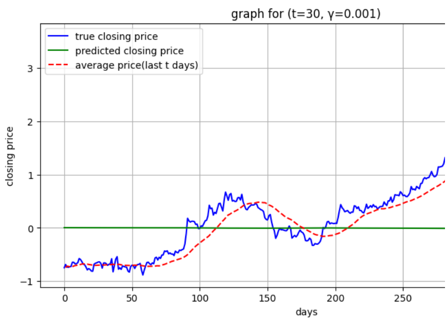
Kernel SVR (t=30, $\gamma = 0.01$)



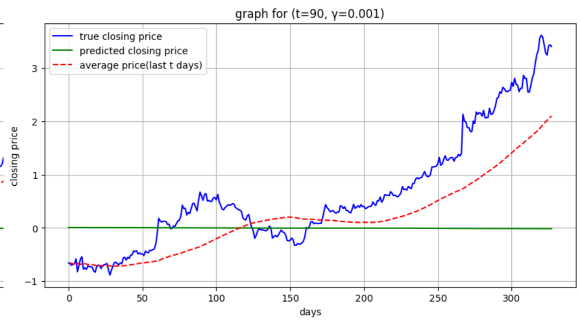
Kernel SVR (t=90, $\gamma = 0.01$)



Kernel SVR (t=7, $\gamma = 0.001$)



Kernel SVR (t=30, $\gamma = 0.001$)



Kernel SVR (t=90, $\gamma = 0.001$)