Maxie Castaneda
Professor David
COMP 347
9 May 2022

Support Vector Machines (SVM)'s Application in Economics Currency Exchange Forecasting

Forecasting is important in the study of Economics. Forecasting can predict inflation, economic growth, house prices, exchange rates and population growth, all of which can facilitate policy makers in regards to making decisions that affect the future of the country. Traders, financial managers and policy makers can benefit from knowing future trends of exchange rates in order to facilitate decisions when it comes to trading, liabilities or planning public policy (Yıldıran). Different algorithms have been used to predict currency exchange values, the three most common are Auto-Regressive Integrated Moving Average (ARIMA) framework, Artificial Neural Networks (ANNs), and Support Vector Machine (SVM). When it comes to financial time-series data prediction, recent studies have shown that the best forecasting algorithm is SVM Regression, or SVR for short.

Auto- Regressive Integrated Moving Average (ARIMA)

The Box and Jenkins' ARIMA is one of the most used algorithms for forecasting time-series data. The ARIMA algorithm combines three equations. The first equation is the AutoRegressive equation: AR(p), a regression model with lagged observations of y that are determined by a p-th time in the past used as a predictor.

$$\hat{y}_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

In the equation, p is the number of lagged observations, ε is the standard error or the "noise" of the data at that particular time and c is a constant and φs are parameters. The model has a dependent relationship with an observation and a number of lagged observations. The second

equation is the integration equation: I(*d*) which is the difference of raw observations that is taken *d* number of times until the model becomes stationary. The model must be stationary because it eliminates the possibility of the model being biased given different sample spaces.

$$By_t = y_{t-1} \text{ where B is the backshift operator}$$

$$\text{The d-th order difference is: } y'_t = (1 - B)^d y_t$$

The last equation is the moving average: MA(*q*), a model that takes in the past forecasting errors.

$$\hat{y}_t = c + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q}$$

The ε is the standard error or the "noise" at time *t*, *c* is a constant, and *θ*s are parameters. Combining them you get the standard notation: ARIMA(*p, d, q*).

$$\hat{y}_t = c + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + \ldots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

Where *p* is the number of lag observations in the model, *d* is the number of times that the raw differences are calculated and *q* is the size of the moving average window. When it comes to execution, there are three steps. The first step is identification, where you assess whether the model is stationary or not and then define the parameters of an ARIMA model for the data, if the model is not stationary, you correct it. Estimation is where you use the data to train your model and lastly diagnostic testing and where you check the accuracy of your model through checking whether or not the fit of the model and the residuals (Brownlee).

The ARIMA has disadvantages for predicting these types of models. To begin, determining the parameters *p* and *q* is difficult and a lot of trial and error exist if you do not have experience. The ARIMA model is also not good for long term forecasting since they cannot predict turning points (Understanding Arima Models for Machine Learning). In the study "Forecasting USDTRY rate by ARIMA method," Cenk created a short-term and a long-term prediction model using the ARIMA framework to predict USD to Turkish Lira, and they

concluded that the ARIMA method was more effective for short-term predictions, (Yıldıran). In another study titled “Comparing ANN Based Models with ARIMA for Prediction of Forex Rates" Kamruzzaman also concluded that “the quality of forecast with ARIMA model deteriorates with the increase of the number of periods for the forecasting (/testing) phase,” (Kamruzzaman) meaning that the ARIMA model is a better fit for short-term forecasting, (Kamruzzaman). The ARIMA model also suffers from being a linear estimate, if the data does not have a linear tendency, it will skew and yield less accurate results (Nayak). Kamruzzaman also concluded that when the ARIMA model was compared to an Artificial Neural Network (ANN) model, the ANN model outperformed the ARIMA model based on their different performance metrics.

Artificial Neural Networks (ANN)

An ANN is composed of many layers. The input layer is where the data is received, the output layer is where the problem solution is obtained. Between the output and the input layer, we have the hidden layer, which is where the processing occurs. There are individual nodes, or neurons, that exist in each of the layers that signal each other as information is processed. Each neuron is assigned a certain weight and these weights determine the output of the ANN.

The inputs are defined as: $x_i = (x_1, x_2, \ldots, x_n)$

The weights can be defined as: $W_j = (w_{1j}, w_{2j}, \ldots, w_{nj})$

Where w_j represents the strength of connection between the input and the processing unit b_j

The processing unit b_j is defined as: $b_j = f(\sum_{i=1}^{n} x_i w_{ij} - w_{0j} - \varphi_j) = f(W_j)$

b_j is the nonlinear activation function. The activation function is what tracks and prevents the output values getting to large values, values which stun the ANN model. The training is

accomplished by giving your ANN a set of inputs and attaching desired outcomes. The learning algorithm is called backpropagation (Alamili). The training process starts with assigning different weights where the weights are adjusted and the validity of the model is determined by the backpropagation learning algorithm. The training process is divided into two steps. Step one is the propagation phase, where the training pattern is put into the model and step two is when the propagation output activations are given. Then the propagation outputs are put into the model where the "deltas," of the output weight are given. Each weight is updated according to the delta, the input activation and the learning rate.

ANN has been widely used when it comes to forecasting and has been successful, however, it does fall short in some aspects. ANN suffers from multiple local minimums, never finding the global minimum (Nayak). They depend a lot on input space and are prone to overfitting or underfitting. ANNs require a lot of input parameters, and there is no structured way to determine the optimal parameters (Alamili). This means that you will have to experiment with the different parameters, which is costly and does not guarantee perfect results.

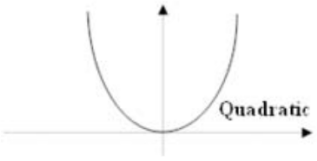
Support Vector Machines Regression (SVR)

SVM is a new technique, proven to be more efficient not only than the ARIMA model but the ANN model as well. SVM Regression, known as SVR, is a classification algorithm which requires the SVM to be trained with a large data set that is split into training and testing sets to classify (Alamili). SVR can be applied to time-series prediction. The cost function is defined as:

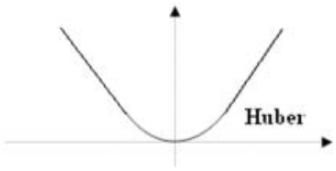
$$q(C) = \frac{1}{2}\,||w||^2 + C\frac{1}{n}\sum_{i=1}^{n} \textit{loss function}$$

Where q is a function of the soft margin C. C is a constant that is bigger than 0 and is meant to be a regularizing term determining the trade-off between the training error and the model's flatness. The loss function can vary, but the most common loss functions for SVM (Regression) are linear, quadratic, and Huber (Awad), Laplace and ε-insensitive (Wang).

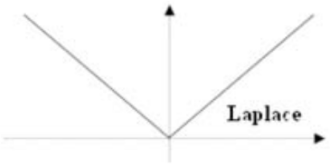
Linear (Awad)



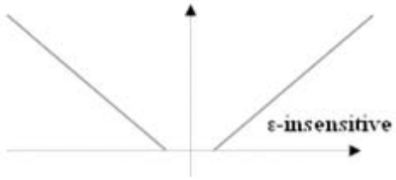
Quadratic (Wang)



Huber (Wang)



Huber (Wang)



ε - insensitive (Wang)

For the explanation we will use the ε-sensitive loss function as follows:

$$q(C) = \frac{1}{2}||w||^2 + C\frac{1}{n}\sum_{i=1}^{n} L_\varepsilon(y_i, y_i^*)$$

$$L_\varepsilon(y_i y_i^*) = \begin{cases} |y_i^* - , y_i| - \varepsilon & \text{if } |y_i^* - , y_i| \geq \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

The constrained optimization problem defined as:

$$\text{minimize } w, \xi^{(*)} \qquad q(C) = \frac{1}{2}||w||^2 + C\frac{1}{n}\sum_{i=1}^{n}(\xi_{i+}\xi_i^*)$$

$$\text{Subject to:} \qquad y_i - \langle w. x_i\rangle - b_i \leq \varepsilon + \xi_i,$$

$$-y_i - \langle w. x_i\rangle - b_i \leq \varepsilon + \xi_i^*,$$

$$\xi^{(*)} \geq 0.$$

The lagrange theory is used to obtain a dual formation of the optimization problem where *w* combines the training data:

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

Where α_i^* and α_i are lagrange multipliers that are associated with a specific instance of x_i. The asterisk means if it is below or above the regression line. One advantage of SVM is the use of a "kernel," which assists the algorithm in determining the shape of the hyperplane, can be chosen case by case depending on the data and plays a vital role in the algorithm's success. The optimization problem takes the following form:

$$f(x_i\, \alpha_i, \alpha_i^*) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*)\, K(x, x_i) + b$$

Where $K(x, x_i)$ represents the kernel of choice. The kernels that have been used in work for currency forecasting are:

Linear: $K(x_i, x_j) = \langle x_i . x_j \rangle$

Polynomial: $K(x_i, x_j) = (\langle x_i . x_j \rangle + 1)^d$, d = degree

Radial Basis: $K(x_i, x_j) = exp\,(- ||x_i - x_j||^d / 2\sigma^2)$, σ = width

Spline: $K(x_i, x_j) = 1 + \langle x_i . x_j \rangle + (\tfrac{1}{2}) \langle x_i . x_j \rangle \min(\langle x_i . x_j \rangle) - (\tfrac{1}{6}) \min \langle x_i . x_j \rangle$

The parameters α_i, α_i^* are obtained by maximizing the following equation:

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

$$\text{Subject to:} \quad \sum_{i=1}^{n} (\alpha_i^* - \alpha_i) = 0$$

$$0 \leq \alpha_i^*, \alpha_j^* \leq C$$

SVM kernels implicitly contain nonlinear transformations, if the parameters are well chosen (C and ε) then the classification results turn out good and it allows for a unique solution, which is convenient if the problem is complex (Nayak). The key insight in this problem is to choose the correct parameters for C and ε. The increase of ε increases the bias, decreases the variance and leads to a lower model complexity. The increase of C decreases the bias, increases the variance and thus leads to a higher model complexity (Alamili).

Applications of SVM to Forecast Foriegn Exchange Market

In Kamruzzaman's study, "SVM' based models for predicting foreign exchange rates," they investigated and tested different types of kernels on different currencies to determine which was the best kernel for currency forecasting. They tested linear, polynomial, radial basis and spline. They chose to compare the United States Dollar (USD), British Pound (GBP), Japanese Yen (JPY), Singapore dollar (SGD), New Zealand dollar (NZD) and Swiss Franc (CHF) against the Australian Dollar using data from January 1991 to July 2002. They used Normalized Mean Square Error (NSME), Mean Absolute Error (MAE), to measure deviation between actual values and forecast values. They used Directional (DS), Correct Up trend (CU), and Correct Down trend (CD) to determine if the direction of prediction was correct. All five were used to evaluate the kernel's effectiveness. When choosing their regularization parameter C, they experimented with values ranging from 0.1 to 10^3, they chose ε to be 10^{-3}. They used time delay moving average to fix the irregularities in the exchange rates. The moving average values of past weeks were used to build the SVM model in order to predict values for the following week. They had six parameters used to assist the support vector: "MA5 (moving average on 5 days), MA10, MA20, MA60 and MA120 and x_i, last week's closing rates," (Kamruzzaman). The period

predicted is i+1. They discovered that there is not a single kernel that was best for all the data, it depended on the data's historical trend.

When looking at their measures of error (NSME and MAE), for JPY, who had the most fluctuating data benefited from using a spline kernel. For USD and GBP the linear kernel was the best since there was a continuous trend without many sharp increases or decreases. For the SGD, NZD, and CHF the polynomial kernel was the most suitable. When measuring the direction of the trends, the linear kernel was the least effective for every currency. With respect to directional change, the radial basis kernel was the best for JYP, NZD and CHF, while the polynomial kernel was the best for USD and SGD. The ideal SVM model is one that "minimizes the prediction errors while maximizing the coverage of directional matching," (Kamruzzaman). In terms of choosing their *C* parameter, they compared the NMSE, MAE and CS errors against different values of *C* with the linear kernel on the JYP data; they concluded that choosing a *C* above 50 yielded the same results.

In Alamili's research "Exchange Rate Predictions using Support Vector Machines. A comparison with Artificial Neural Networks," they compared an ANN model with an SVM model, where they concluded that the SVM model was better. While using an SVM model to predict the currency exchange rate, they tested various kernels on the EUR/USD data and used the NMSE to measure the performance. They tested values of *C* ranging from 0.001 and 1. They found that *C* was proportional to the model's complexity, the more *C* increases, the more accurate the prediction on the training set is and the poorer the prediction is on the validation set. Thus the higher levels of *C* overfit the data, while the lower levels of *C* underfit the data. They also tested different values for ε and determined that the size of ε had little to no effect on the complexity of the model thus left it at $\varepsilon = 0.001$, a value that is often used in literature. In terms

of the type of kernel that was most effective, they tested four different kernels, linear, polynomial, radial basis function, and the sigmoid function. Their kernels included an additional parameter γ which is the inverse width where the more γ increase, the better the prediction is on the training set and the poorer the prediction is for the validation set. Larger values of γ lead to an over fit, smaller values lead to an under fit. For the radial basis function, they tested various values of γ against the NSME and concluded that an appropriate range was $0.01 \leq \gamma \leq 100$. The first kernel they tested was the linear kernel, which proved to not be suitable for predicting the EUR/ USD. They used the standard value of ε = 0.001, and even with varying the values of *C*, the results were not promising, thus they eliminated the linear kernel from being tested. Their main focus was on the radial basis kernel:

$$\text{Radial Basis: } K(x_i, x_j) = e^{-\gamma||xi - xj||2}, \gamma > 0$$

Where they tested different values for *C* ranging from 0.01 to 1, γ ranging from 0.01 to 100 for a window size of 3, 7, and 15. The best results were found through the use of a window size = 3 where the *C* is between 0.1 and 1 and γ is between 0.55 and 1.78.

Overall, in their comparison of SVM to ANN, SVM is easier to use and can be more efficient in some cases. The radial basis kernel they used only had three parameters *C*, ε, and γ. Neural networks on the other hand "require a larger number of controlling parameters, including the number of hidden layers, the number of hidden nodes, learning rate, the momentum term, the number of iterations, transfer functions, and weights initialization methods," (Alamili). While it is difficult for both models to obtain the optimal parameters, since SVM has less parameters it is more efficient. They also found that "training an SVM is equivalent to solving a linearly constraint quadratic programming problem, and that the solution of the SVM is unique, optimal and global. This is however not the case for the neural network, which may not converge to

global solutions," (Alamili). In terms of which model was best for predicting EUR/USD, it would depend on the goal while using the model because the NSME and hidrate values vary.

When implementing a Support Vector Regression for time-series prediction, the first thing to consider is the loss function you are going to use. In Alamili and Ahmed's work, they used the ε-sensitive loss function. In Li's research "Financial market forecasting using a two-step kernel learning method for the support vector regression," they used the Laplace loss function since it is "more robust to outliers than the quadratic loss function and it requires fewer parameters than the ε-insensitive and the Huber loss functions," (Wang). After choosing your loss function you choose your kernel, which all depends on your data's historic trend. Once you choose your kernel, you can implement your model and test different values for your parameters C, ε and γ (if you decide to include it) and assess it against your evaluation metrics.

Implementation via Gradient Descent

In the paper "Large-scale support vector regression with budgeted stochastic gradient descent," Li introduced a Stochastic Gradient Descent (SGD) method for SVR using the e-insensitive loss function, which is the function that is widely used in SVR algorithms. The SGD algorithm that was used with the SVR cost function was Pegasos.

Pegasos Algorithm

For SVM the problem is defined as:

$$min(w) \quad \frac{\lambda}{2}||w||^2 + \frac{1}{2}\sum_{(x,y)\epsilon S} L(w;(x,y))$$

Where $L(w;(x,y)) = max\{0,\ 1 - y\langle w, x\rangle\}$

We get an initial vector w and chose a random training sample then replace the objective from the SVM problem yielding the following:

$$min(w, i_t) \qquad \frac{\lambda}{2}||w||^2 + L(w; (x_{i_t}, y_{i_t}))$$

Where the subgradient descent is:

$$\nabla_t = \lambda w_t - 1\,[y_{i_t} \langle w_t,\ x_{i_t} \rangle < 1]\, x_{i_t} y_{i_t}$$

Where $1\,[y_{i_t} \langle w_t,\ x_{i_t} \rangle < 1]$ is the indicator function which takes a value of its argument and is true or zero otherwise.

The update for w_t

$$w_{t+1} \leftarrow w_t - \eta_t \nabla_t$$

Where η_t is a step size of $\frac{1}{\lambda t}$

<u>Pegasos Algorithm for Linear SGD SVR</u>

For SVR:

minimize w: $P(w) = \frac{1}{2} w^T w + \frac{C}{l} \sum_{i=1}^{l} L(w, x_i, y_i)$

where $L(w, x_i, y_i) = max(|w^T x_i - y_i| - \varepsilon,\ 0)$

Initialize your weight vector w and update your w vector until the iteration number is reached. The update:

$w_{t+1} = w_t - \eta_t \nabla_w P_t(w_t)$, η_t = The learning rate

Where $P_t(w_t) = ½\ w^T w + C * L(w, x_t, y_t)$

$\nabla_w P_t(w_t)$ is the sub gradient descent and applying the Pegasos algorithm, we can rewrite the update function as:

$$w_{t+1} = (1 - \eta_t) w_t + \beta_t x_t$$

Where $\beta_t = 0$ *if* $|w^T x_t - y_t| \leq \varepsilon$

$-C$ *if* $|w^T x_t - y_t| > \varepsilon$

C *if* $|w^T x_t - y_t| < -\varepsilon$

For this project, I will implement linear SVR via SGD from the algorithm given above. The pegasos algorithm has an ideal learning rate, however since the sample size was smaller, I decided in my implementation I am going to test various learning rates. There will be various values for the *C* parameter tested along with the learning rates, these will be tested with the default and commonly used ε value of 1.0e-4. However, once the idea learning rate and *C* parameter is chosen, different values for ε will also be tested.The model is going to be trained with the Mexico-US exchange rate history from 1994-2020. The model will then be tested with data from 2021.

Function

```
def SVR_linear_sgdNoBatch(x, y, eps, w, c_val, learning_rate, iterations):
    l = len(x)
    previous_cost = None
    previous_error = None
    count_iters = 0
    reached = False
    for i in range(iterations):
        current_error = rmse(x, w, y)
        if i > 1 and previous_error <= current_error:
            break
        for j in range(l):
            b = B_vals(x[j], y[j], w, eps, c_val)
            w = np.multiply((1 - learning_rate), w) + (b * x[j])
            loss = e_loss(w, x[j], y[j], eps)
            current_cost = SVR_cost(w, c_val, l, loss)
            if i > 1 and previous_cost <= current_cost:
                break
            previous_cost = current_cost
        previous_error = current_error
        # print("The Error:", current_error)
        count_iters += 1
    if iterations == count_iters:
        reached == True
    return w, reached, current_error, current_cost
```

This is the main SVR function. We can see that the update parameter is in the j loop.

```
def B_vals(x, y, w, eps, c_val):
    wT = w.reshape(-1, 1)
    xR = x.reshape(1, -1)
    if np.linalg.norm(np.abs(np.dot(wT, xR) - y)) <= eps:
        return 0
    if np.linalg.norm(np.dot(wT, xR) - y) > eps:
        return -c_val
    if np.linalg.norm(np.dot(wT, xR) - y) < -eps:
        return c_val
```

In order to update the w vector we need to update our Beta.

```
def rmse(x, w, y):
    pred = []
    for i in range(len(x)):
        pred.append(each_data(x[i], w))
    inside = y - pred
    total = np.linalg.norm(inside) ** 2
    return np.sqrt(total / len(x))
```

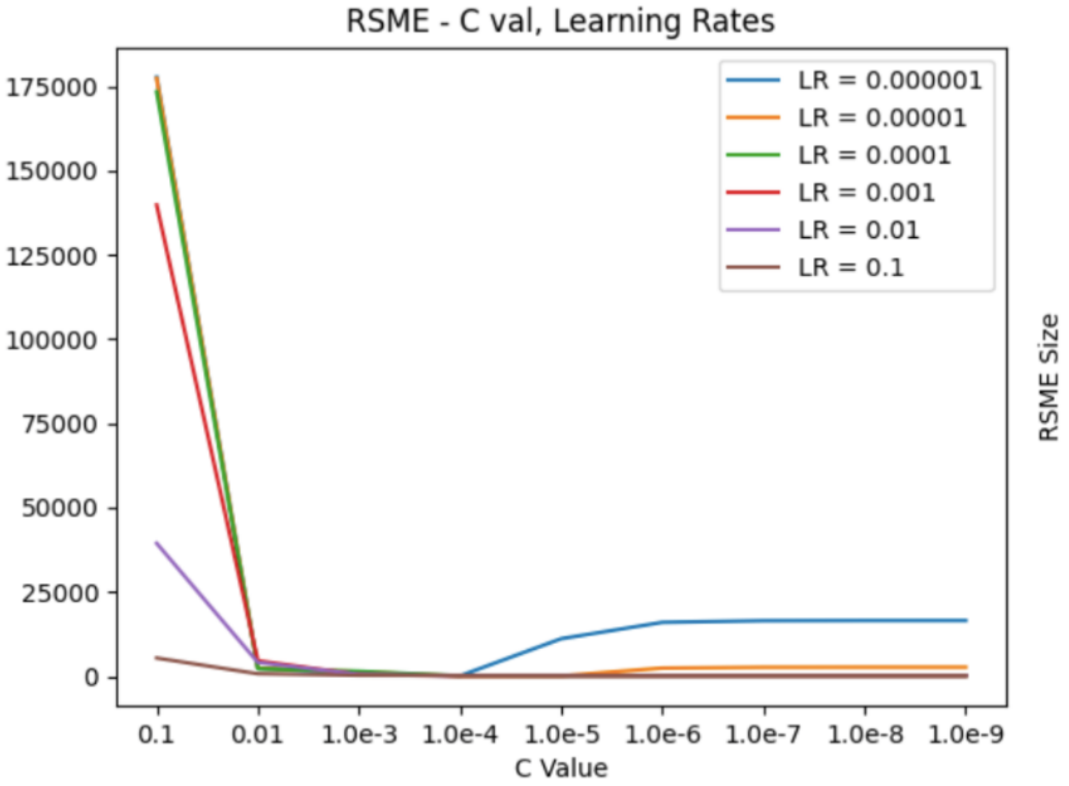
Within our algorithm we have stop functions that check if the RMSE error begins to get larger.

```
def SVR_cost(w, c, l, loss):
    w2 = w.reshape(1, -1)
    norm = np.linalg.norm(w2)
    return .5 * (norm * norm) + (c / l) * loss
```

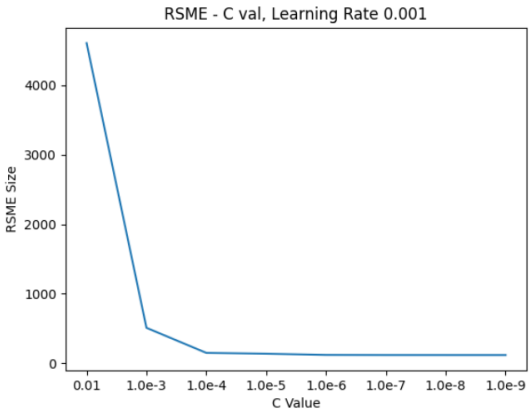
```
def e_loss(w, x, y, eps):
    wT = w.reshape(-1, 1)
    xR = x.reshape(1, -1)
    return np.max(np.linalg.norm(np.abs(np.dot(wT, xR)) - y) - eps, 0)
```

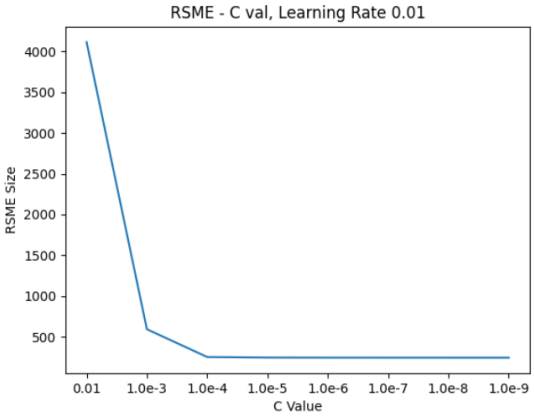
As well as checking the cost of the currency w vector.

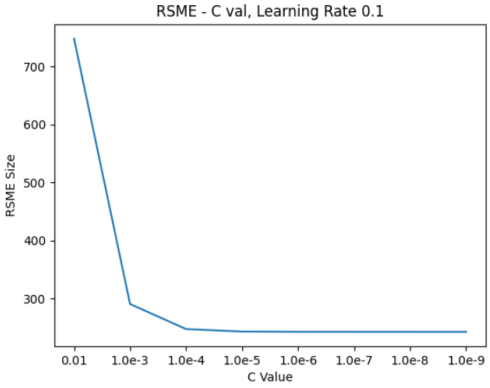
Applying this function, we can test the following values: C ranging from 1.0e-9 to 1 and a learning rate of 1.0e-5 - 0.1 (a standard epsilon of e= 0.0001 was used, ranging values will be tested at the end).

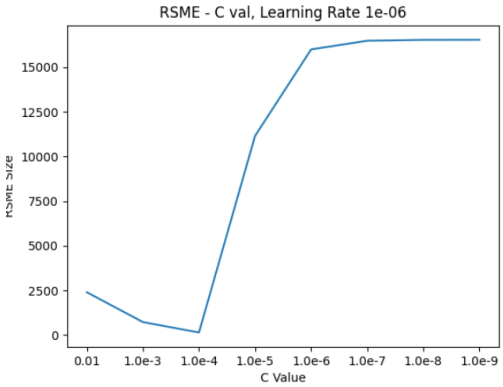


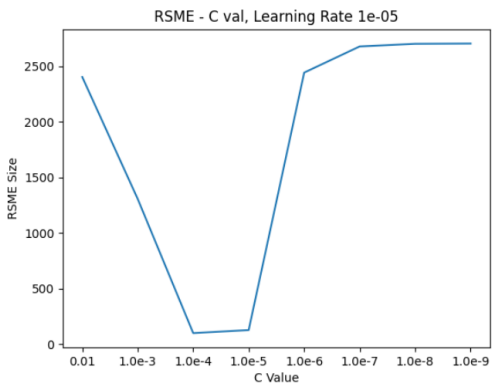
At first glance, it looks as though a higher learning rate gives us smaller RMSE errors. We instead now individually see each learning rate against the C value. For this currency exchange market, lower values of C yield the best results.

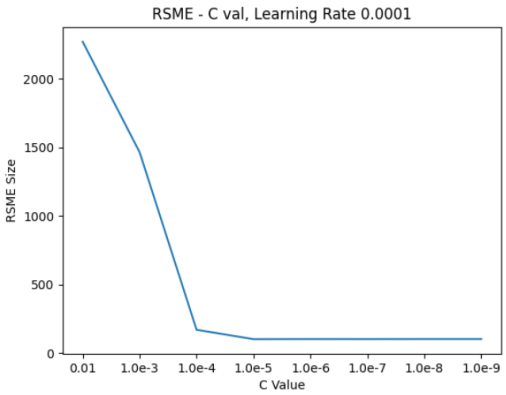




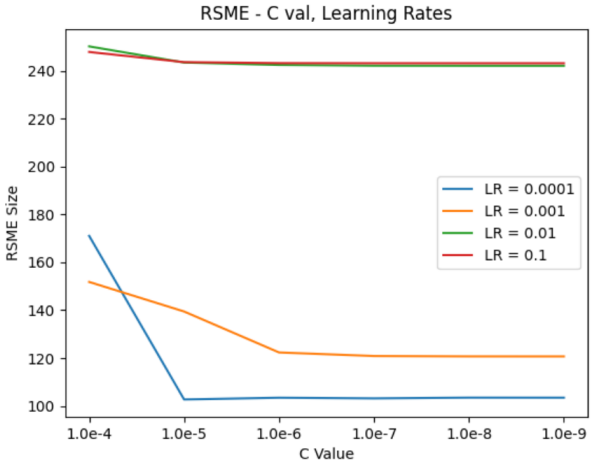


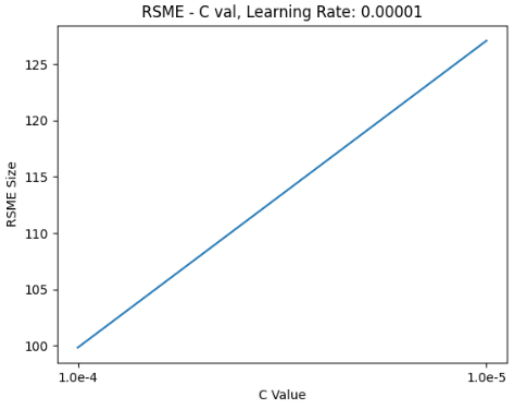


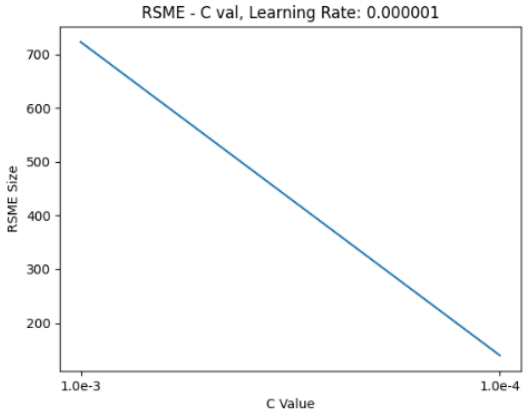




We need to see the values closely.







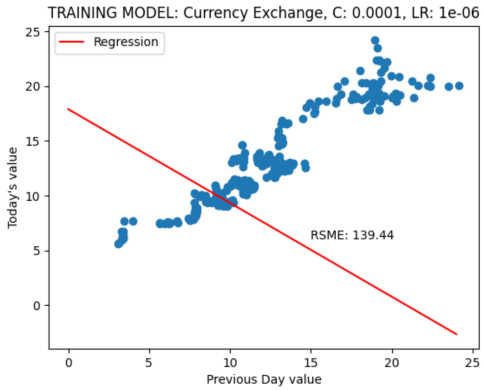
Given this, a learning rate between 0.001 and 0.000001 are the lowest RMSE errors. So we will test the following values:

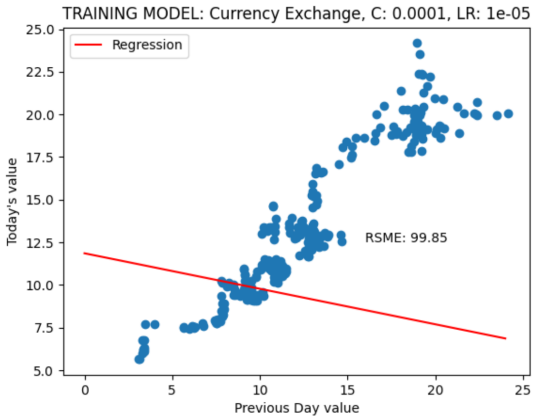
LR 0.001 with C: 0.00001- 0.0000001

LR 0.0001 with C: 0.00001- 0.0000001

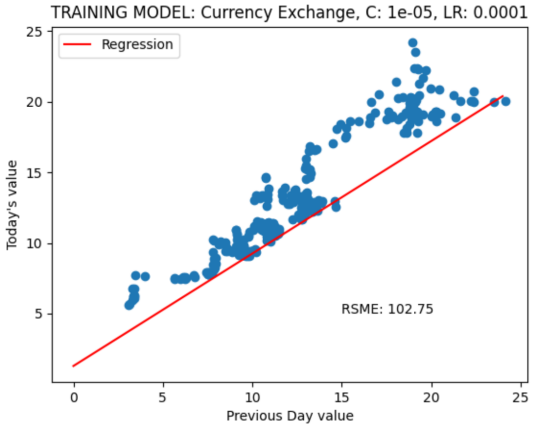
LR 0.00001 with C: 0.0001- 0.000001

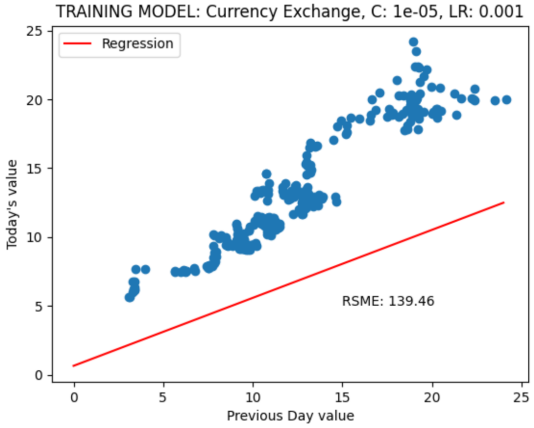
LR 0.000001 with C: 0.001- 0.0001

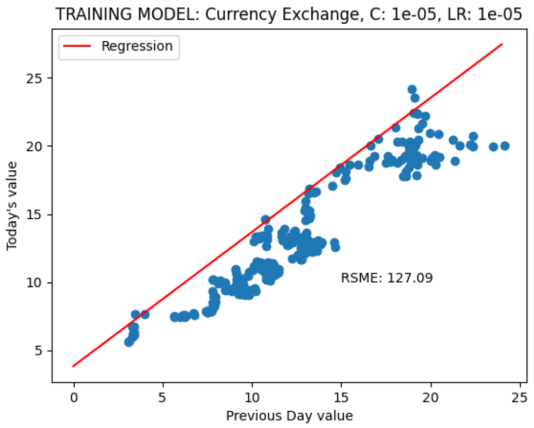


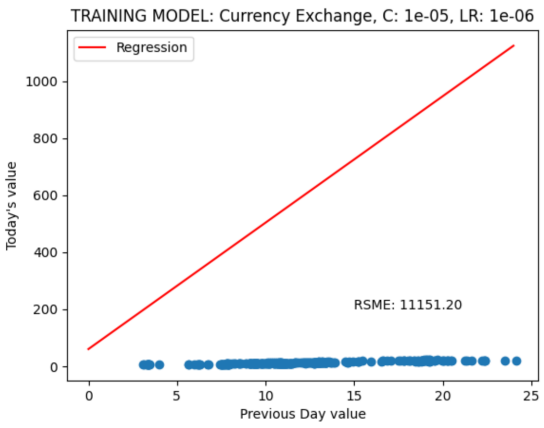


For C = 0.001, both considered learning rates were eliminated.

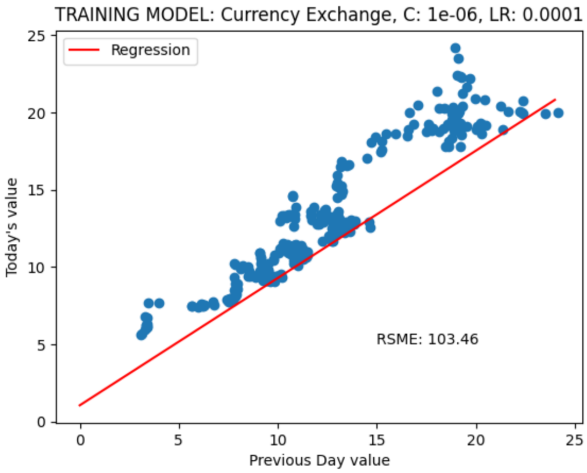


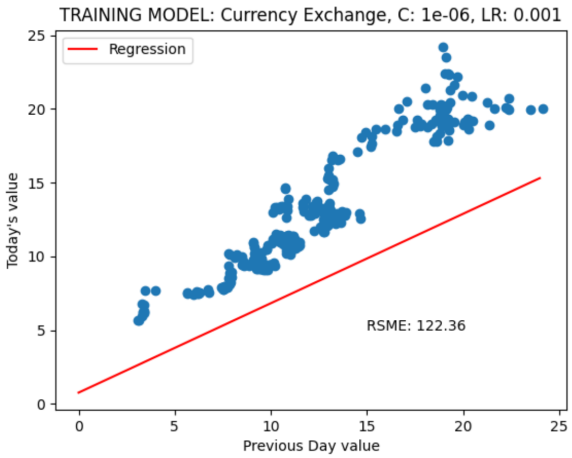


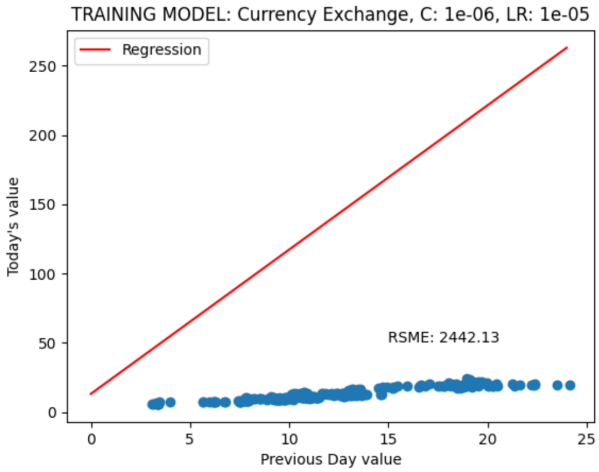




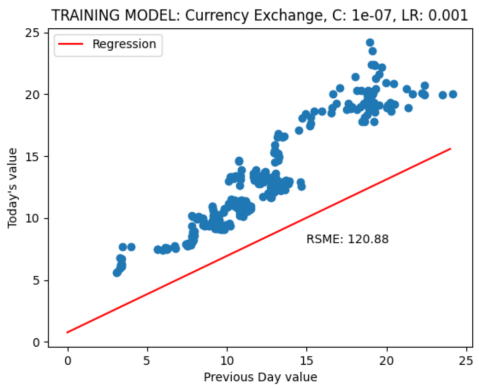
For C = 0.00001, the correct trend began to appear, further enforcing that smaller values of C are best.

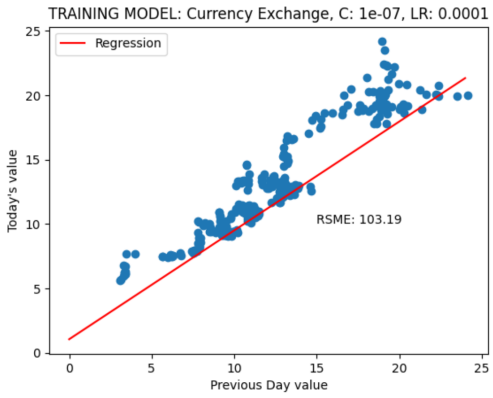






For C = 0.000001. There was a mass improvement.





For C= 0.0000001, the correct trend was captured.

We will iterate through various learning rates for our best C values in order to find the best learning rate. The algorithm is as follows:

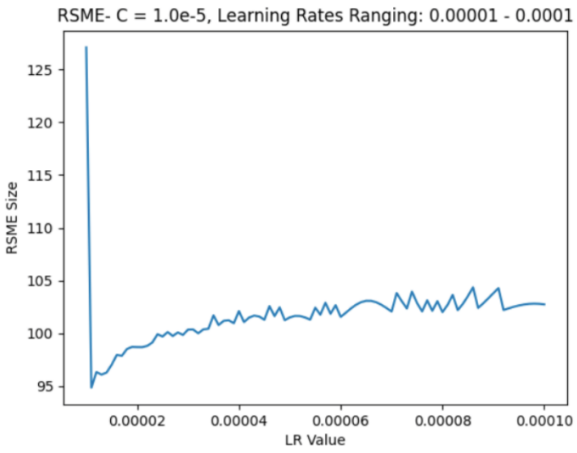
```
def iterate_LR_values(x, y, eps, w, c_val, min_LR, increment, max_LR, iters):
    count = 0
    rsme = []
    cost = []
    reached = []
    W_vects = []
    LR = []
    while min_LR < max_LR:
        LR.append(min_LR)
        W, reach, CE, CC = SVR_linear_sgdNoBatch(x, y, eps, w, c_val, min_LR, iters)
        rsme.append(CE)
        cost.append(CC)
        reached.append(reach)
        W_vects.append(W)
        min_LR = min_LR + increment
        count += 1
    return rsme, cost, reached, W_vects, count, LR
```

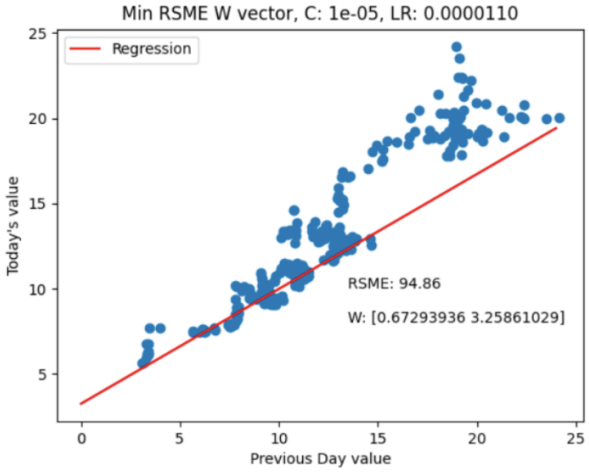
We run the function above to rest a range of learning rates, and then extract the lowest RMSE value and its corresponding Learning rate.

```
min_error = min(error)
min_index_error = error.index(min_error)

SV_error = support_vectors[min_index_error]
SV_LR_error = learning_rate_array[min_index_error]

min_error = format(min_error, ".2f")
SV_LR_error = format(SV_LR_error, ".7f")
```

The above algorithm was used in the following steps:

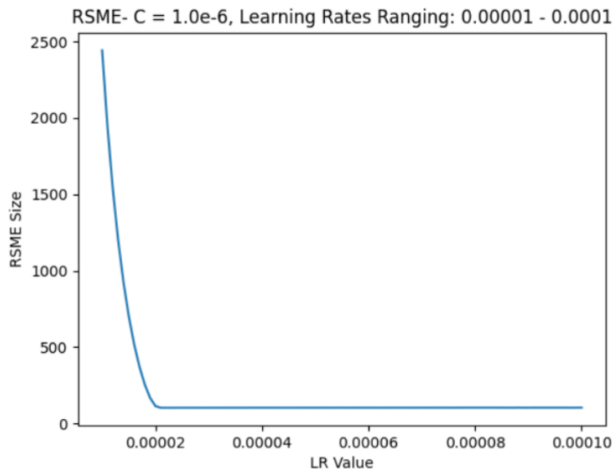
C = 0.00001 with a learning rate within the range of 0.0001 - 0.00001

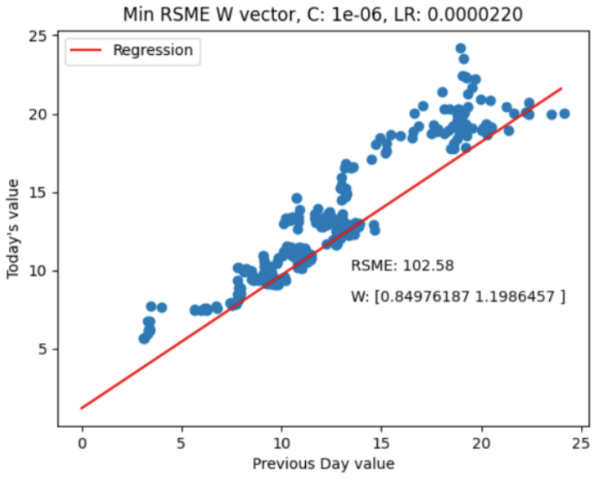




Our ideal learning rate was 0.0000110

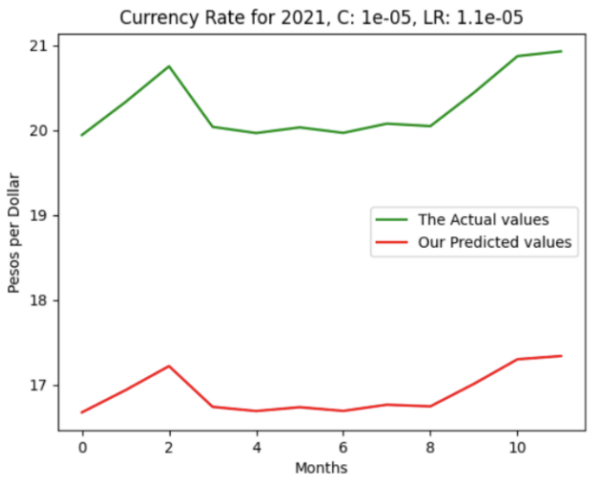
C = 0.000001 with a learning rate within the range of 0.0001 - 0.00001

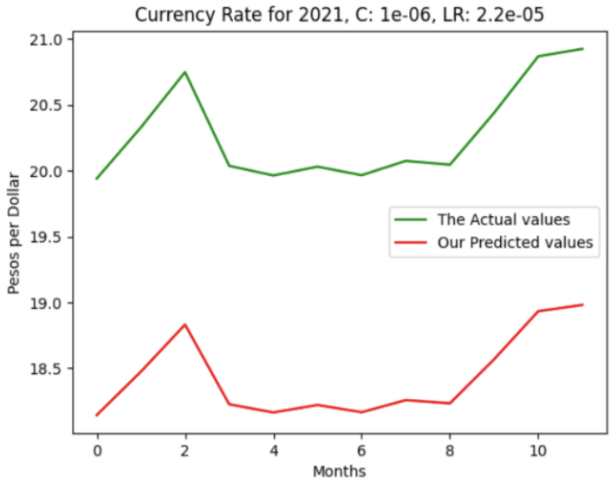




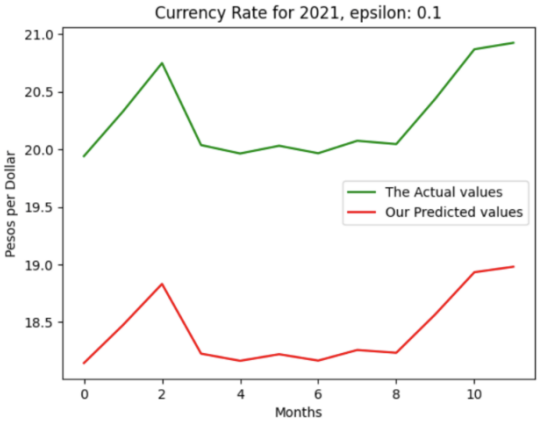
Our ideal learning rate was 0.0000220

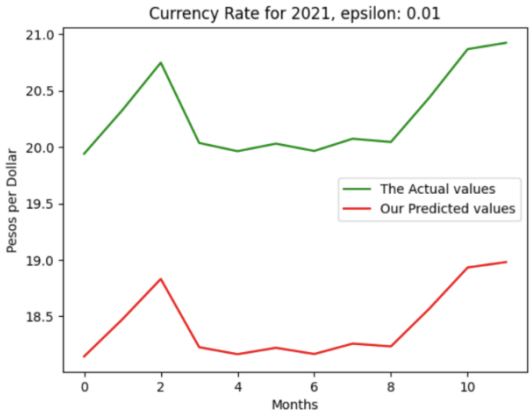
We took these ideal C and learning rates and put them into our testing data:

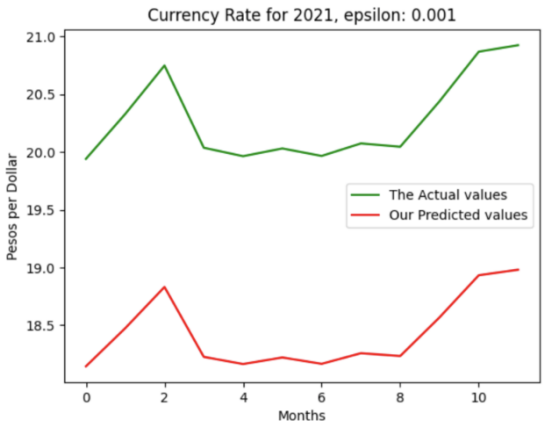


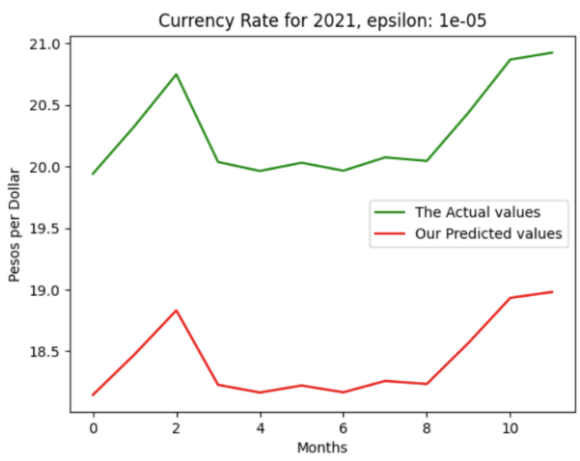


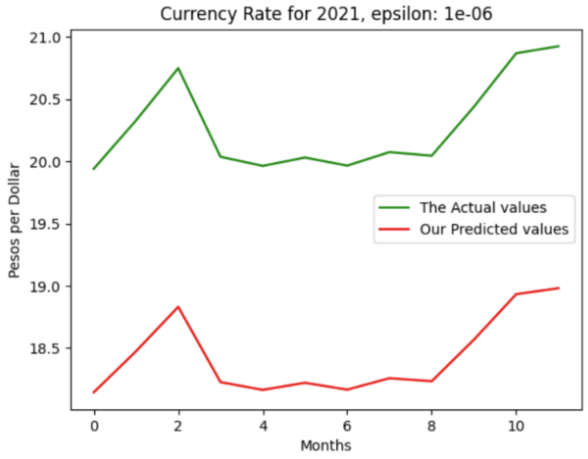
Our results seemed to be a lower estimate. The next step was to test a range of epsilon values.







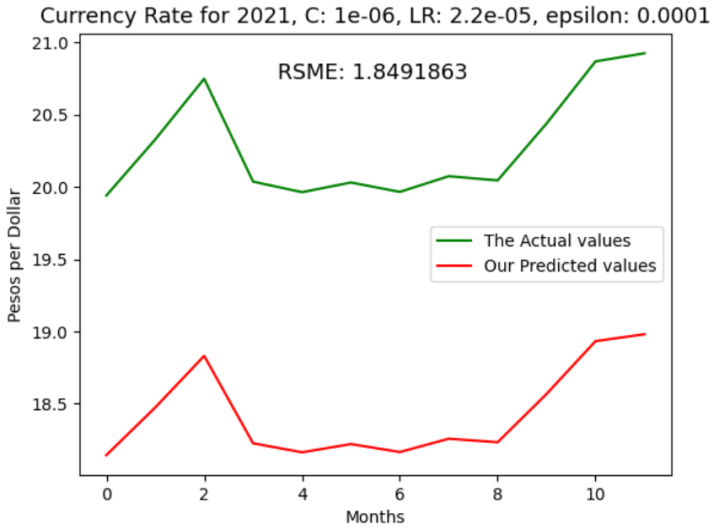




Changing the epsilon values did not change our results.

Conclusion

Using the following parameters: C = 0.000001, LR = 0.0000220, Eps = 0.0001 we got a RSME of 1.849 and our model was an underestimate of the real data.



Some things to keep in mind are that perhaps a linear kernel was not the best for our data, additionally it is difficult to get these parameter's perfect as said by past researchers. To improve this model, different kernels should be considered and perhaps an additional parameter Lambda to account for the small fluctuations in the data.

Bibliography

Alamili, Mohamad. *Exchange Rate Prediction Using Support Vector Machines*. https://d1rkab7tlqy5f1.cloudfront.net/TBM/Over%20faculteit/Afdelingen/Engineering%20Systems%20and%20Services/People/Professors%20emeriti/Jan%20van%20den%20Berg/MasterPhdThesis/M-Alamili-MSc-Final.pdf.

Awad, M., Khanna, R. (2015). Support Vector Regression. In: Efficient Learning Machines. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-5990-9_4

Brownlee, Jason. "A Gentle Introduction to the Box-Jenkins Method for Time Series Forecasting." *Machine Learning Mastery*, 14 Aug. 2020, https://machinelearningmastery.com/gentle-introduction-box-jenkins-method-time-series-forecasting/.

Kamruzzaman, J., & Sarker, R.A. (2003). Comparing ANN Based Models with ARIMA for Prediction of Forex Rates.

Kamruzzaman, J., et al. "SVM Based Models for Predicting Foreign Currency Exchange Rates." *Third IEEE International Conference on Data Mining*, https://doi.org/10.1109/icdm.2003.1250976.

Nayak, Rudra Kalyan, et al. "An Optimized SVM-k-NN Currency Exchange Forecasting Model for Indian Currency Market." *Neural Computing and Applications*, vol. 31, no. 7, 2017, pp. 2995–3021., https://doi.org/10.1007/s00521-017-3248-5.

"Understanding Arima Models for Machine Learning." *Capital One*, https://www.capitalone.com/tech/machine-learning/understanding-arima-models/.

Shalev-Shwartz, Shai, et al. *Pegasos: Primal Estimated Sub-Gradient Solver for SVM*. https://home.ttic.edu/~nati/Publications/PegasosMPB.pdf.

Wang, Li, and Ji Zhu. “Financial Market Forecasting Using a Two-Step Kernel Learning Method for the Support Vector Regression.” *Annals of Operations Research*, vol. 174, no. 1, 2008, pp. 103–120., https://doi.org/10.1007/s10479-008-0357-7.

Xie, Zongxia, and Yingda Li. “Large-Scale Support Vector Regression with Budgeted Stochastic Gradient Descent.” *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 6, 2018, pp. 1529–1541., https://doi.org/10.1007/s13042-018-0832-7.

Yıldıran, Cenk Ufuk, and Abdurrahman Fettahoğlu. “Forecasting USDTRY Rate by Arima Method.” *Cogent Economics & Finance*, vol. 5, no. 1, 2017, p. 1335968., https://doi.org/10.1080/23322039.2017.1335968.