STAT LEARN Final Project

An alternative approach to forecast the volatility of multiscale and high-dimensional market data

Qingmin Shi
168007883

Abstract

Traditional methods for volatility forecast of multiscale and high-dimensional data like foreign-exchange and stock market volatility have both advantages and disadvantages which have been identified. In my project, I apply the Support Vector Machine (SVM) as a complimentary volatility method which is capable dealing of such type of data. SVM-based models may extract extra information of time series data and handle the long memory effect very well. Our Support Vector Machine for Regression (SVR) model has better result than the common GARCH (1, 1) model. The predictions are closer to the historical data and the error is lower. In addition, I test different kernels to see the performance difference. For my data, rbf kernel has an overall better performance than linear and polynomial kernels. I conclude that SVM-based model may be applied more frequently in the emerging field of high-frequency finance and in multivariate models for portfolio risk management.

1. Data

I apply daily China / U.S. foreign exchange rate (DEXCHUS) data from 1/3/2006 ~ 12/2/2016. After clearing data, there are 2574 effective data point in total. All the data is obtained from the Federal Reserve Bank of St. Louis official website. I separate the data as 60% percent for training group and rest 40% percent for testing purpose. I plot several graphs of the original data to show some basic statics including the FX rate plot (Figure 1), daily return plot (Figure 2) and volatility plot (Figure 3). And the first few rows of our input data matrix are showed as well (Figure 4).

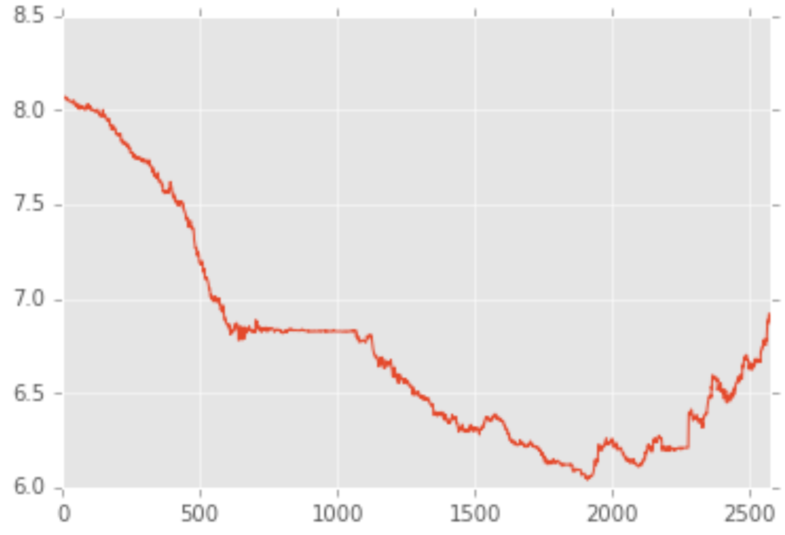


Figure 1

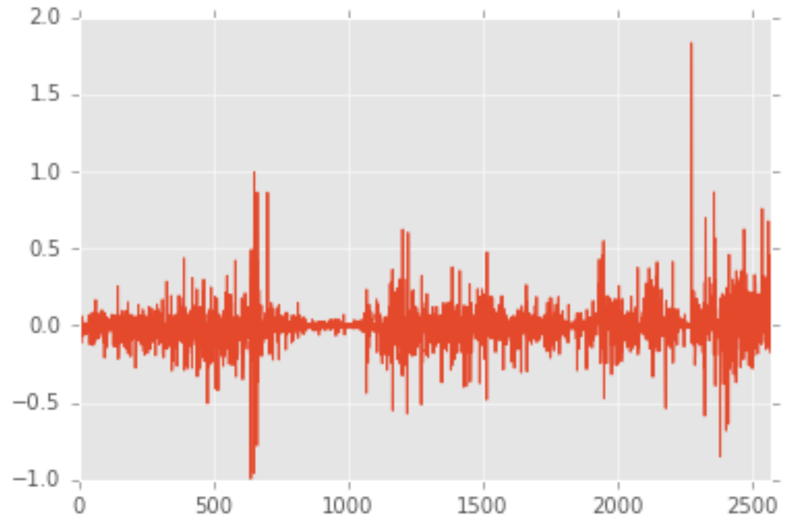


Figure 2

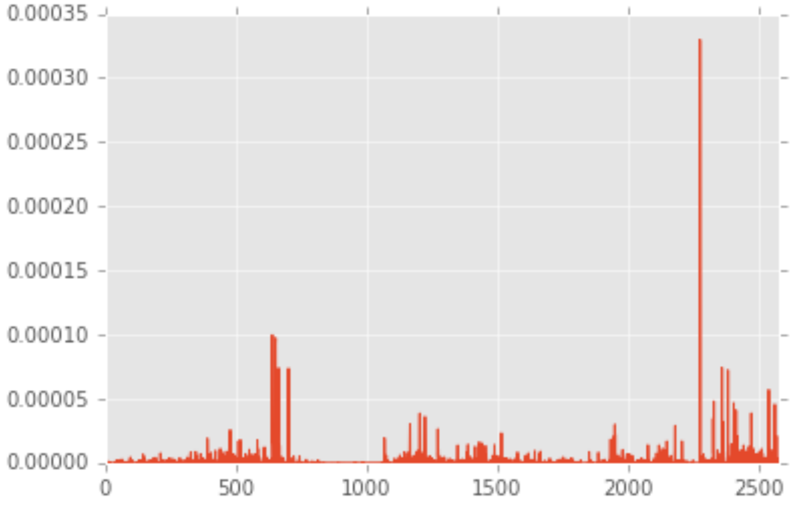
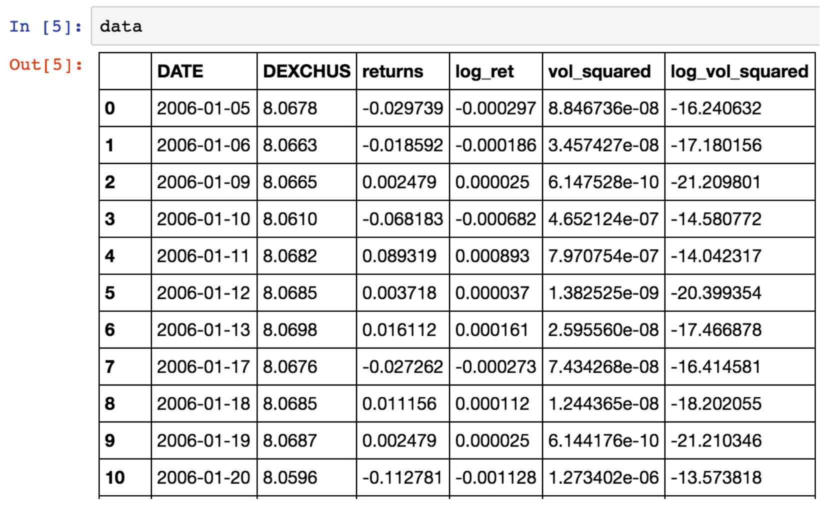


Figure 3



```
In [5]: data
Out[5]:
```

	DATE	DEXCHUS	returns	log_ret	vol_squared	log_vol_squared
0	2006-01-05	8.0678	-0.029739	-0.000297	8.846736e-08	-16.240632
1	2006-01-06	8.0663	-0.018592	-0.000186	3.457427e-08	-17.180156
2	2006-01-09	8.0665	0.002479	0.000025	6.147528e-10	-21.209801
3	2006-01-10	8.0610	-0.068183	-0.000682	4.652124e-07	-14.580772
4	2006-01-11	8.0682	0.089319	0.000893	7.970754e-07	-14.042317
5	2006-01-12	8.0685	0.003718	0.000037	1.382525e-09	-20.399354
6	2006-01-13	8.0698	0.016112	0.000161	2.595560e-08	-17.466878
7	2006-01-17	8.0676	-0.027262	-0.000273	7.434268e-08	-16.414581
8	2006-01-18	8.0685	0.011156	0.000112	1.244365e-08	-18.202055
9	2006-01-19	8.0687	0.002479	0.000025	6.144176e-10	-21.210346
10	2006-01-20	8.0596	-0.112781	-0.001128	1.273402e-06	-13.573818

Figure 4

2. Models and Methodology

My forecasting is mainly based on two models, SVR model and ARMAxGARCH model. SVR is my main purpose of the project and ARMAxGARCH is applied as a comparable test. In my SVR model, the first step is to select the best kernel for our data. To amplifier the results, I selecting a relatively large moving window size, which is 50. And I train the model with my training dataset. The testing dataset is prepared for test my model's sufficiency and effectiveness. One of my biggest concerns is the starting point. For the specialty of the time series data, I think the separation of data is very critical and which does lead to almost opposite conclusion in my aspect. With my first try, I only leave last 20% data as my testing group. The results with different moving window size are different but all optimistic. After presentation, I modified the testing group as last 40% of the original data. And the testing results are much more disappointed than ever. I conclude this due to the memory effect and the sample size. My data size may not be big enough to establish a firm model analysis and the memory effect of the time series data can add prediction errors if I forecast with the previous predictions. After all, I will show specific output and graphs later.

2.1 SVR Model

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated. In my project, I test both linear and non-linear SVR models with different kernels.

Linear SVR: $$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \cdot \langle x_i, x \rangle + b$$

Non-linear SVR: $$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \cdot \langle \varphi(x_i), \varphi(x) \rangle + b$$

Polynomial

$$k(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_i . \mathrm{x}_j)^d$$

Gaussian Radial Basis function

$$k(\mathrm{x}_i, \mathrm{x}_j) = \exp\left(- \frac{\|\mathrm{x}_i - \mathrm{x}_j\|^2}{2\sigma^2} \right)$$

Kernel functions:

2.2 GARCH Model

In his seminal paper, Engle (1982) introduced the class of autoregressive conditional heteroskedastic (ARCH) models to capture the volatility prevalent in a time series. He applied this model to estimate the variance of inflation in the United Kingdom. The original ARCH model was later generalized by Bollerslev (1986) and this is known as the generalized ARCH (GARCH) model. Since its introduction, this class of models has been applied in countless empirical studies, especially those involving financial market data, due to their success in modeling persistence and volatility which are typically present in financial variables. An AR (*k*)-GARCH (*p, q*) model for a stationary time series $\{y_t\}$ is represented as

$$y_t = \Phi_0 + \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \cdots + \Phi_k y_{t-k} + \varepsilon_t$$

where $\varepsilon_t \mid \Psi_{t-1} \sim N(0,h_t)$, $\Psi_{t-1} = \{y_{t-1}, y_{t-2}, \ldots\}$ is the information set at $t-1$ and

$$h_t = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \cdots + \alpha_q \varepsilon_{t-q}^2 + \beta_1 h_{t-1} + \cdots + \beta_p h_{t-p} . \qquad (3.3)$$

All the roots of k $\Phi(B) = 1 - \Phi_1 B - \ldots - \Phi_k B^k$ lie outside the unit circle so that the time series $\{y_t\}$ is stationary. To ensure positivity of the conditional variance, h_t, the following conditions on the parameters in (3.3) namely, $\alpha_0 > 0$ and $\alpha_i \geq 0$ for $i = 1, 2, \ldots, q$ and $\beta_i \geq 0$ for $i = 1, 2, \ldots, p$, are required. Weaker sufficient conditions are also available.

3. Empirical Findings and Analysis

3.1 Estimated Models

In this section, I present two estimated models. As stated earlier, the estimation of the models was carried out based on the data covering the period 1/3/2006 to 12/2/2016.

3.1.1 SVR Model

First, I input the training dataset and set the moving window as 50 to get the results which are easier to tell the difference. After controlling everything else as the same, I apply different kernel functions one by one, including linear kernel, polynomial kernel and rbf kernel to predict the last 100 data points within training group (Figure 5). The results show clearly that SVR with rbf kernel has the best performance since they are closest to the real data.

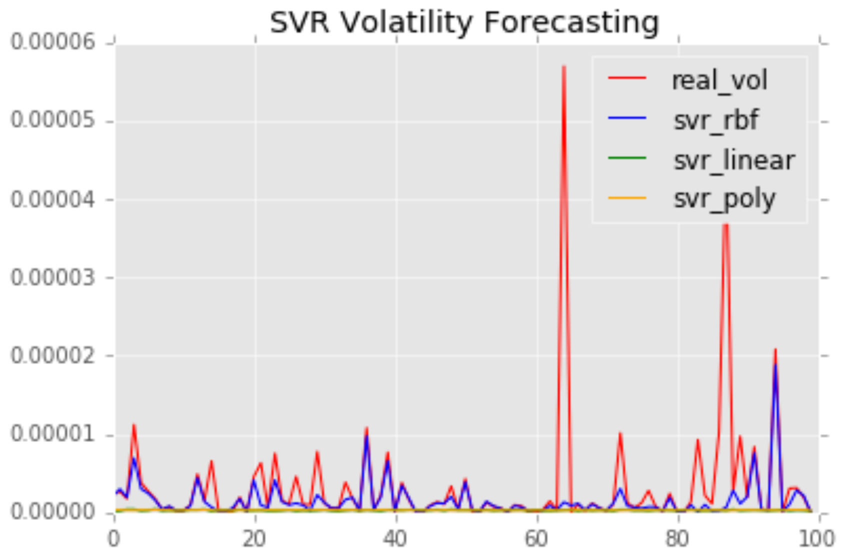


Figure 5

After picking the rbf kernel, I try to figure out the best size of moving window period. Within the training dataset, I predict the last 100 hundred data point with window size from 10 to 40, increasing by 10 (Figure 6~9). The results show that better performance with larger window size, which makes sense since more data leads to more information and better prediction.

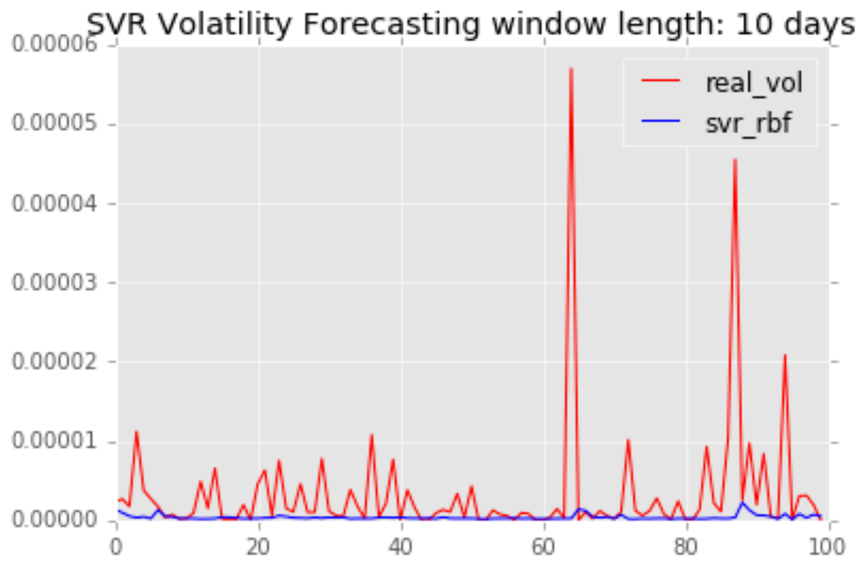


Figure 6

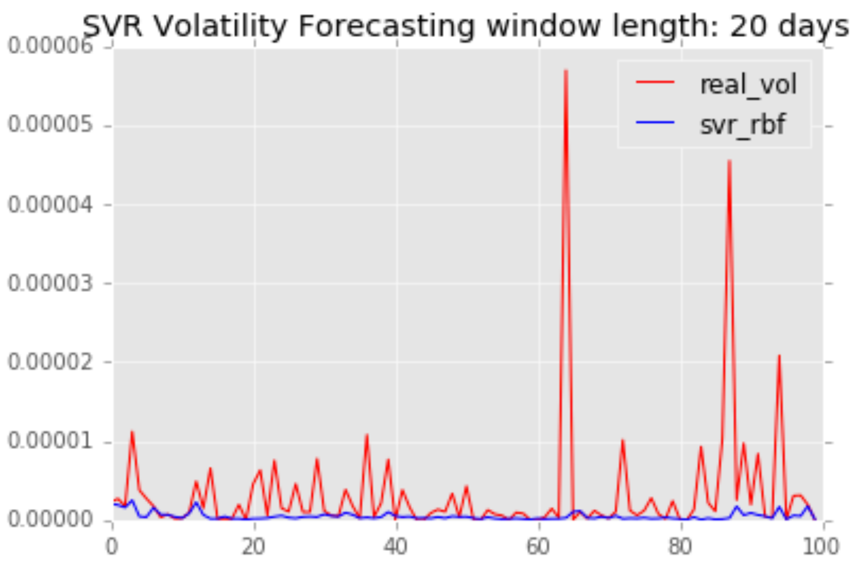


Figure 7

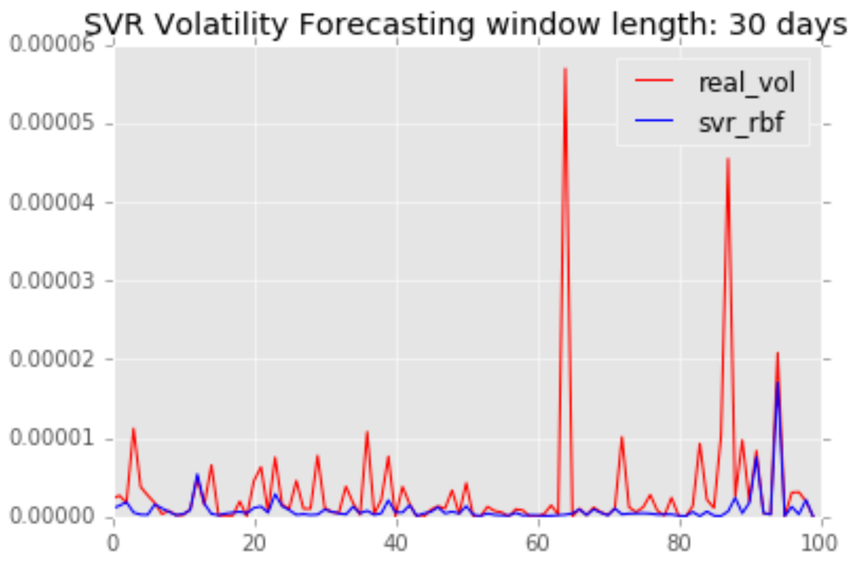


Figure 8

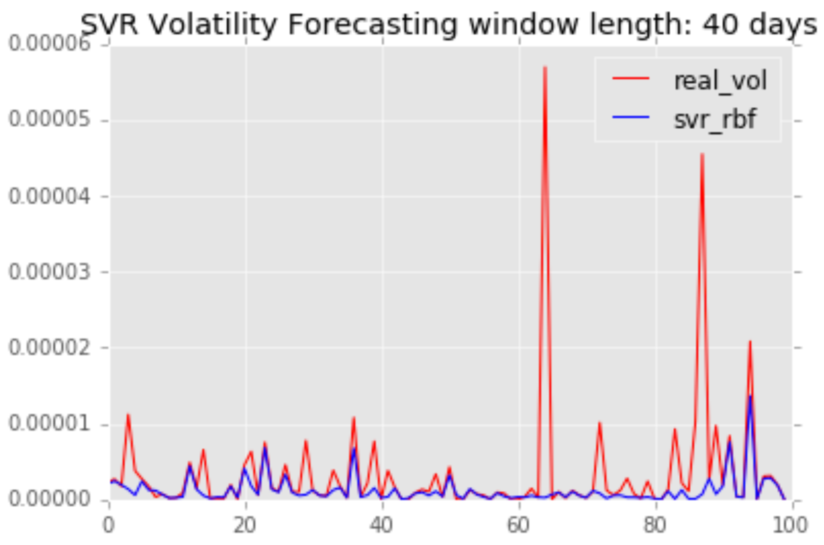


Figure 9

Thus, in next step, I will fill in the same model with testing data and try to figure out the performance difference and any indices for overfitting. The size of my predictions equals the size of the testing dataset, which is 1029. However, the prediction is quite different from the previous output (Figure 10~13). I also check the sum of squared errors of different models. And the SSE keeps decreasing from 2.57481368579 for window size 10 to 0.0082528249095 for window size 40. And the prediction output is showed below (Figure 14).

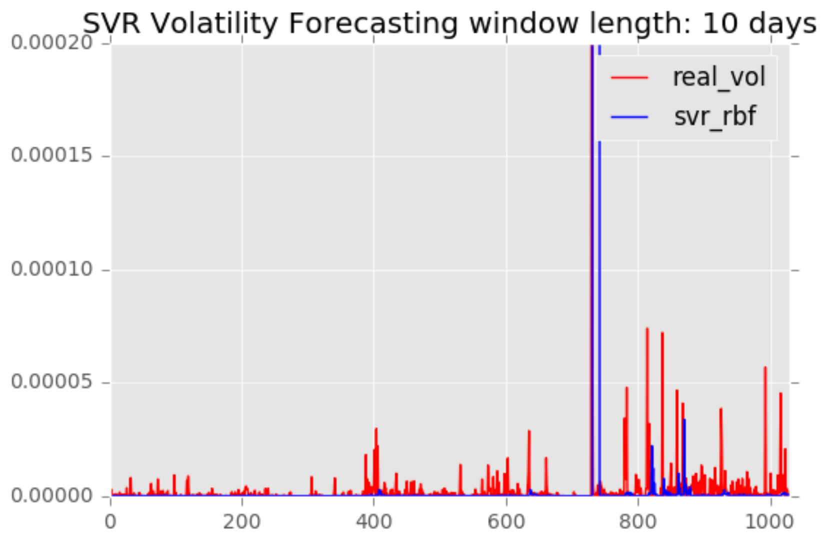


Figure 10

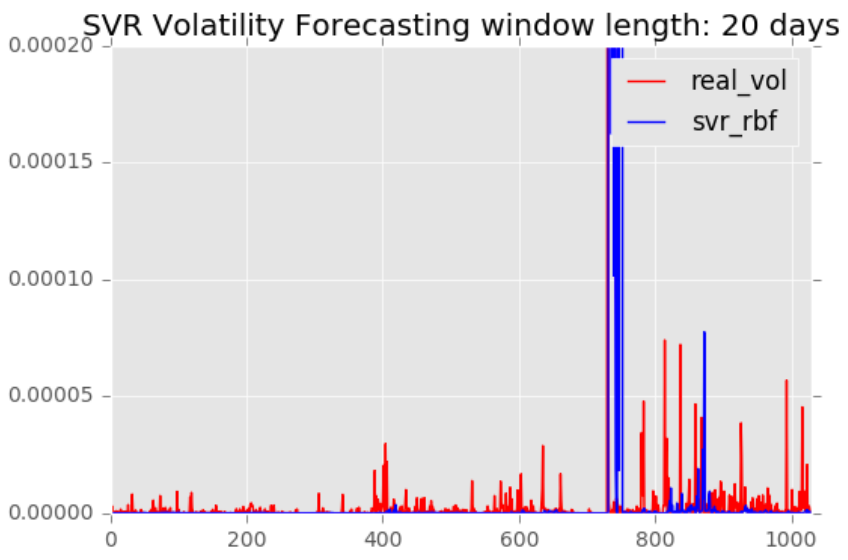


Figure11

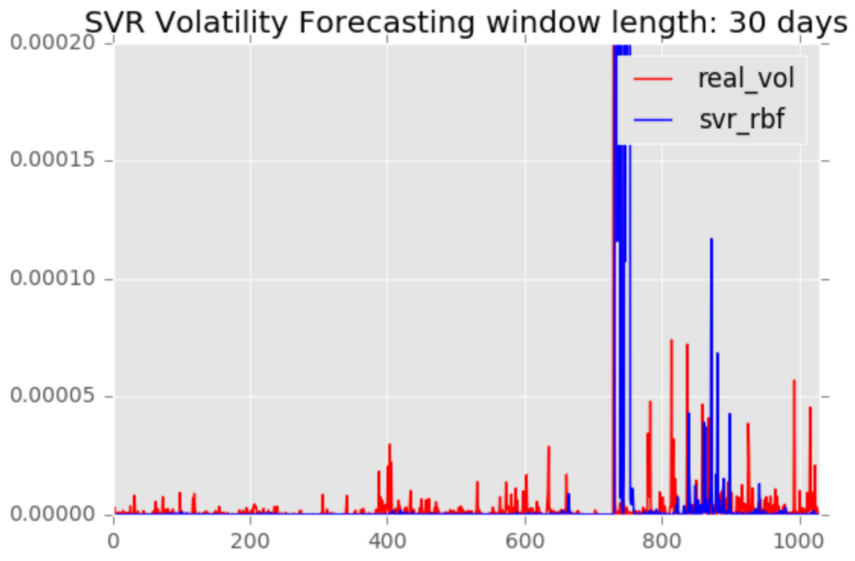


Figure 12

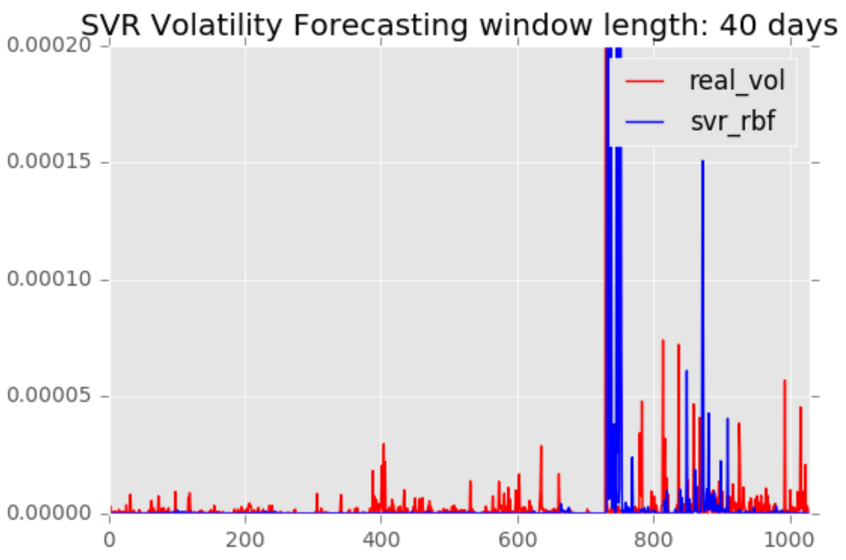


Figure 13

```
array([[  2.04092847e-04,  -1.56998195e-04,   1.03664388e-03, ...,
         -6.35152516e-05,   6.35152516e-05,  -1.30125701e-03],
       [ -2.82601189e-04,   2.04092847e-04,  -1.56998195e-04, ...,
          2.34738169e-03,  -6.35152516e-05,   6.35152516e-05],
       [  2.66903218e-04,  -2.82601189e-04,   2.04092847e-04, ...,
         -1.12542116e-03,   2.34738169e-03,  -6.35152516e-05],
       ...,
       [ -1.75069313e-03,  -2.74624020e-04,   4.56274556e-03, ...,
          6.00420312e-04,  -1.83015607e-03,  -1.01863512e-03],
       [ -1.76827017e-03,  -1.75069313e-03,  -2.74624020e-04, ...,
          2.06868712e-03,   6.00420312e-04,  -1.83015607e-03],
       [ -1.39362729e-03,  -1.76827017e-03,  -1.75069313e-03, ...,
          1.49738706e-04,   2.06868712e-03,   6.00420312e-04]])
```

Figure 14

*3.1.2 ARMA*x*GARCH Model*

Based on the ACF, PACF and EACF plot, I finally choose the ARMA(3, 0)xGARCH(1,1) model, the predictions are not even in the same scale of the prediction of my SVR model with training data (Figure 15).

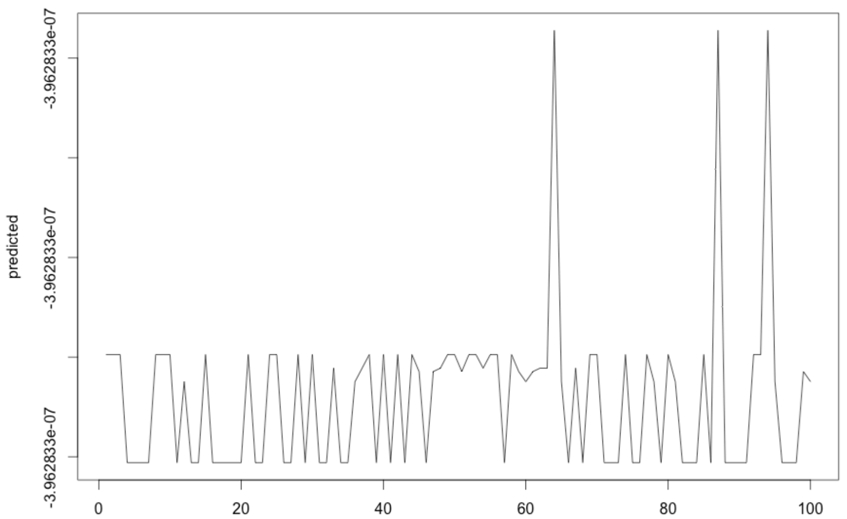


Figure 15

3.2 Forecast Performance

The performance of different models is quite obvious, with SVR with rbf kernel, I get the predictions which partially close to the real data. However, with traditional GARCH model, the prediction which at the scale of 10^{-7} are far less than the real data, which are at the scale of 10^{-5}.

4. Concluding Remarks

The statics refer to the choice of larger window size, but from the graph (Figure 10~13), I prefer the window size of 10 since its predictions around day-count 750 and 870 are less extreme than other plots. We can tell from the plot that with larger window size, it is more likely to be overfit. Although my prediction is not good enough, it is still far better than my results from GARCH model. Thus, I conclude that SVR model is more capable for multiscale and high-dimensional market data like foreign exchange rate, which has long-memory, and I can apply it without strict assumptions and approximations which may be required by other models.

Code:

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

# import arch
import numpy as np
import pandas as pd
from sklearn.svm import SVR
import matplotlib.pyplot as plt


data = pd.read_csv('DEXCHUS.csv')
data = data[data['DEXCHUS'] != '.']
data.DEXCHUS = data.DEXCHUS.astype(float)
data['returns'] = 100 * data.DEXCHUS.pct_change().dropna()
data['log_ret'] = np.log(data.DEXCHUS) - np.log(data.DEXCHUS.shift(1))
data = data[data.log_ret!=0]
data['vol_squared'] = data['log_ret']**2
data['log_vol_squared'] = np.log(data['log_ret']**2)
data = data[1:]
data = data.reset_index(drop=True)


data.DEXCHUS.plot()
data.returns.plot()
data.vol_squared.plot()
'''
returns = data.returns.dropna().as_matrix()
from arch.univariate import arch_model
am = arch_model(returns)
res = am.fit()
res.summary()
fig = res.plot()
'''

# set n as moving window period
n = 50
X = np.zeros((data.shape[0]-n,n))
for i in range(n,data.shape[0]):
    for j in range(n):
        X[i-n][j] = data.log_ret[i-j-1]
y = np.zeros(data.shape[0]-n)
for i in range(data.shape[0]-n):
    y[i] = data.log_vol_squared[i+n]
```

```
svr_rbf = SVR(kernel='rbf',C=1e3,gamma=250)
y_rbf = svr_rbf.fit(X,y).predict(X)
predicted_vol_rbf = np.exp(y_rbf)

svr_linear = SVR(kernel='linear',C=1e3,gamma=250)
y_linear = svr_linear.fit(X,y).predict(X)
predicted_vol_linear = np.exp(y_linear)

svr_poly = SVR(kernel = 'poly',C=1e3,gamma=250)
y_poly = svr_poly.fit(X,y).predict(X)
predicted_vol_poly = np.exp(y_poly)

real_vol = data.vol_squared[n:].as_matrix()
xaxis = range(150-n)

plt.style.use('ggplot')
plt.plot(xaxis,real_vol[data.shape[0]-
150:],color='r',label='real_vol')
plt.plot(xaxis,predicted_vol_rbf[data.shape[0]-
150:],color='b',label='svr_rbf')
plt.plot(xaxis,predicted_vol_linear[data.shape[0]-
150:],color='g',label='svr_linear')
plt.plot(xaxis,predicted_vol_poly[data.shape[0]-
150:],color='orange',label='svr_poly')
plt.title('SVR Volatility Forecasting')
plt.legend()
plt.show()

def SVR_with_moving_windows(n):
    X = np.zeros((data.shape[0]-n,n))
    for i in range(n,data.shape[0]):
        for j in range(n):
            X[i-n][j] = data.log_ret[i-j-1]
    y = np.zeros(data.shape[0]-n)
    for i in range(data.shape[0]-n):
        y[i] = data.log_vol_squared[i+n]

    svr_rbf = SVR(kernel='rbf',C=1e3,gamma=250)
    y_rbf = svr_rbf.fit(X,y).predict(X)
    predicted_vol_rbf = np.exp(y_rbf)

    real_vol = data.vol_squared[n:].as_matrix()
    xaxis = range(100)

    plt.style.use('ggplot')
    plt.plot(xaxis,real_vol[data.shape[0]-100-
n:],color='r',label='real_vol')
    plt.plot(xaxis,predicted_vol_rbf[data.shape[0]-100-
n:],color='b',label='svr_rbf')
    plt.title('SVR Volatility Forecasting window length: %d days'%(n))
    plt.legend()
    plt.show()

for i in range(10,60,10):
    SVR_with_moving_windows(i)
```