Volatility Analysis of Stocks Price: A Comparison among Several Times Series Models

Guojin Tang **Shuang Gao** **Junrong Zha**
Center for Data Science
New York University

Abstract

Generating volatility forecasts is an issue with great concern in financial area, which has given vent to a lot of literature. It is well known that time series models such as Stochastic Volatility (SV) model, GARCH model, and machine learning techniques are useful tools to model the volatility. Here we applied SV model, GARCH and LSTM models to six stocks to model the volatility of stock price and compared performance of the three models. Our empirical analysis demonstrated that LSTM performed the best, then SV is better than GARCH. Our study can help to establish a measurement for different volatility models and provided comparisons between model generated volatility and proxy volatility.

1 Introduction & Related work

So far, lots of attention has been paid to modeling and forecasting the volatility of financial time series since it is an important aspect of many financial decisions (Zhuang and Chan, 2004). For example, investment managers, option traders and the financial managerial bodies are all interested in volatility forecasts to construct less risky portfolios or obtain higher profits (Panait and Slavescu, 2012). Thus, it is valuable to find a good way for volatility analysis.
Various models have been used for volatility analysis such as stochastic volatility (SV) models (Taylor, 1982, 1986), generalized autoregressive conditional heteroskedasticity (GARCH) models (Bollerslev, et al., 1995), etc. Related work of improving SV and GARCH models through hybrid way were published recently (Nkemnole E. B. and Abass O., 2017; Zhuang and Chan, 2004). Nkemnole E. B. and Abass O., 2017 combined HMM and GARCH while Zhuang and Chan, 2004 combined HMM and SV to better model the volatility of stock exchange index. Their success lied on solving the problem of excessive persistence in original SV and GARCH models by introducing Hidden Markov Models to allow for different volatility states (periods with different volatility levels) in time series. However, except for this hybrid way, along with the development of machine learning, state-of-the-art technology such as LSTM (Hochschreiter Schmidhuber, 1997) can also be applied to volatility analysis.
In this project, we intended to use three current popular models (SV, GARCH and LSTM) to conduct the volatility analysis of stock price and compare their performance on volatility modeling. Without true volatility data, we used proxy volatility as benchmark to measure the model performance. Similar to related work, we expected to see the improvement of model performance among the three models. Firstly, we will discuss the data were used. Then we will discuss the details of three models and our modifications on them. Finally, We will present simulated volatility of stock price for six stocks from three models respectively and their RMSEs to the proxy volatility. These results can tell which model is better for volatility analysis and provide insights of model choice.

Preprint. Under review.

2 Problem Definition and Algorithm

2.1 Task

The inputs for our baseline ARIMA model is log returns of the close prices of 6 selected stocks. As we can see from Figure 1, the log returns exhibit non-negligible heteroskedasticity. Such models

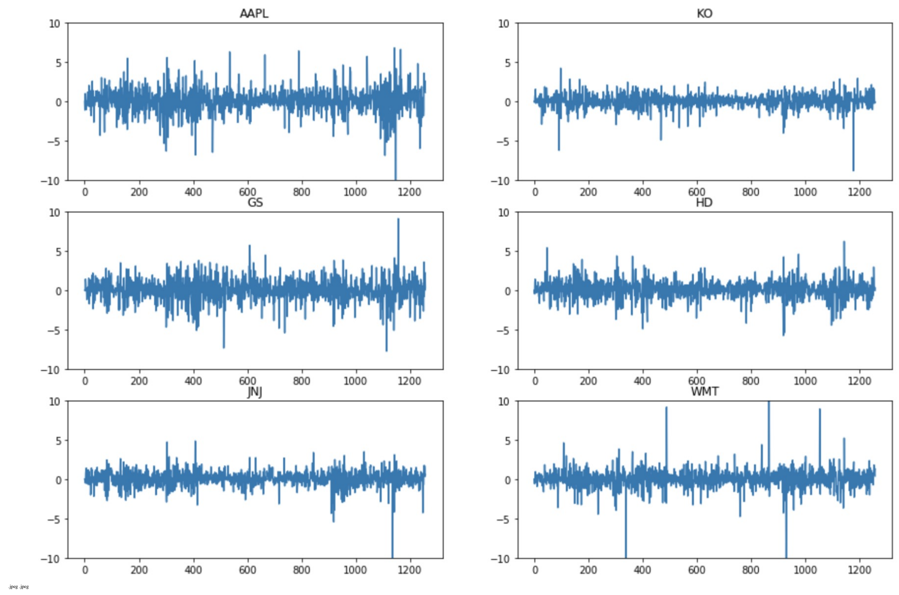

Figure 1: log return of the stock prices for six companies: AAPL,KO,GS,HD,JNJ and WMA. (detailed data description and computation see 4.1 Data)

as ARIMA that work better for constant or periodic variance data would be unsatisfying for capturing the variance structures here. Therefore, we would like to use GARCH, ARIMA-GARCH, SV, SV-t and LSTM models to predict the volatility of the log return of the stock, aiming to see whether they perform better than ARIMA model and compare their results. Inputs were divergent for different models: log returns for GARCH and ARIMA-GARCH, logarithm form of log returns for SV and SV-t, and historical variance, which is also a converted form of log returns, for LSTM. The outputs for all models were the estimated in-sample and the predicted out-of-sample volatilities of the log return of the stock prices, which will be further unified under some simple calculations in order to compare among all the models. (detailed unified proxy calculations see 3.2 Methodology)

2.2 Algorithm

2.2.1 ARIMA Models

ARIMA Introduction ARIMA process works well on stationary data that have autocovariance functions only related to time lags. However, the autocovariance functions for the financial data such as our log return data of the stock prices have much more complicated structures and seldom fluctuate regularly. Therefore, we chose the ARIMA model as a baseline model for comparisons among different models' ability to predict the volatility of log return. Thereby we can see whether other models perform better and how much other models improve.

Hyper Parameters and Model Construction Since we already applied a differencing step in calculating the log return, the d term in ARIMA(p,d,q) should equal to zero, as supported by the above Figure 1 the distribution graphs of the log return. The p and q were determined through comparing ARIMA(p,0,q) models with all possible p,q in range(6). We didn't choose a large range in order to prevent over fitting. The ARIMA models were constructed using function ARIMA() from library statsmodels.tsa.arima_model. We chose AIC as the criterion for best ARIMA model selection because AIC is an estimator of out-of-sample prediction error and our project wants to compare the model's ability to predict the volatility. Also, we used the MLE method for parameter estimation and excluded constant in the model through the fit() function.

```
1 Loop through p = range(6) and q = range(6) to find the arima model with the smallest AIC
  using the following codes:
  arima = ARIMA(train, order = (p,0,q)).fit(method = 'mle', trend = 'nc')
  arima_aic = arima.aic

2 Record the best_arima_model and best_order(p,0,q).
```

In-sample Estimation and Out-of-sample Prediction We used the predict() function to do the in-sample estimation of log return. And we used the recursive prediction method to do the out-of-sample prediction of log return: for each prediction at time t, include the real data up to time t-1 and fit the new model using the same configuration as the above best arima model; then do the next one more prediction using function forecast().

```
1 forecast_in = best_arima_model.predict( start = 0,end = len(train)-1)
2 for i in range(len(train),len(log_return)):
      pred_arima = ARIMA(log_return[0:i],order = best_order).fit(method='mle', trend='nc')
      forecast_out[i-len(train)]=pred_arima.forecast()[0]
```

Proxy Calculation Since we now had the estimated in-sample log return, the predicted out-of-sample log return and the real log return, we used the following equations to calculate the unified proxy in order to compare with other models fairly: $proxy\,unified = \log(log\,return - \overline{log\,return})^2$ (detailed proxy explanation see 3.2 Methodology).

2.2.2 GARCH Models

GARCH Introduction We included GARCH(p,o,q) model to predict the volatility of the log return because it can be considered as an ARMA process applies to the variance and thus it has an emphasis on the variance structure. Using the following example of the equations for GARCH(1,0,1), we will show how to estimate volatility of log return through GARCH model.

$$\begin{aligned} r_t &= \mu + \epsilon_t \\ \epsilon_t &= \sigma_t e_t \\ \sigma_t^2 &= \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2 \\ \text{where } & \omega, e_t \sim N(0,1) \end{aligned}$$

In the above equations, r_t is the input data, μ is the conditional mean, ϵ_t is the residual for the process and σ_t^2 is the conditional variance. So if our input is log return, we can use the following relationship to estimate the volatility of the log return:

$$(log\,return - \overline{log\,return})^2 \sim (r_t - \mu)^2 \sim \epsilon_t^2 \sim \sigma_t^2$$

The leftmost term is our proxy for volatility squared and the rightmost term is our conditional variance from the GARCH model.

Hyper Parameters and Model Construction The p, o and q terms in GARCH(p,o,q) model were determined through comparing GARCH models with all possible p,o,q in range(5). We didn't choose a large range in order to prevent over fitting. The GARCH models were constructed using function arch_model() from the library arch. We again chose AIC as the criterion for best GARCH model selection with the same reason for choosing AIC for the best ARIMA model. Also, if the input log return have a kurtosis more than 3, we used the StudentsT distribution in the function arch_model() and we used the Normal distribution otherwise. The following pseudo codes are the example for the stock AAPL which has a kurtosis of 4.279 for log return.

1 Loop through p = range(6), o = range(6) and q = range(6) to find the GARCH model with the smallest AIC using the following codes:
```
garch = arch_model(train, p, o, q, vol = 'Garch', dist = 'StudentsT').fit(disp = 'off')
garch_aic = garch.aic
```

2 Record the best_garch_model and best_order(p,o,q).

In-sample Estimation and Out-of-sample Prediction We used the square of the conditional volatility results in best_garch_model as the in-sample estimation for the squared volatility of the log return. And we again used the recursive prediction method to do the out-of-sample prediction of volatility: for each prediction at time t, include the real data up to time t-1 and fit the new model using the same configuration as the above best garch model; then do the next one more prediction using function forecast().

1 forecast_in = best_garch_model.conditional_volatility ** 2

2
```
pred_garch=arch_model(log_return, p,o,q,vol='Garch',dist='StudentsT')
for i in range(len(train),len(log_return)):
        result = pred_garch.fit(last_obs = len(train)+i, disp = 'off')
        forecast_out[i] = result.forecast(horizon=1).variance.iloc[i + len(train$_X$) $- 1$]
```

Proxy Calculation For fair comparison, take the log of the forecast_in and forecast_out to compare with the unified proxy using RMSE. (detailed proxy explanation see 3.2 Methodology).

Extension: ARIMA-GARCH After we finished the ARIMA and GARCH model, it came into our mind that what would happen if we took the ARIMA model as a mean model and the GARCH model as the variance model. Therefore, we decided to make a GARCH model on the residuals of the 2.2.1 ARIMA Model. If we input the residuals of ARIMA into the GARCH, the relationship to estimate the volatility of the log return changes to:

$$(log\, return - ARIMA\, fitted\, values - \mu)^2 \sim (ARIMA\, resids - \mu)^2 \sim (r_t - \mu)^2 \sim \epsilon_t^2 \sim \sigma_t^2$$

The leftmost term is log return minus its mean model and some conditional mean of the residuals, which is the same as our proxy for squared volatility of log return. Therefore, we can use the conditional variance to estimate the squared volatility of log return again.

process The whole process of deciding the hyper parameters, constructing models, estimating in-sample volatility, predicting the our-of-sample volatility and calculating the proxy is the same as the above GARCH only model, besides the inputs were the residuals of ARIMA model from 2.2.1.

2.2.3 Stochastic Volatility Models

SV Introduction As an alternative to GARCH family, Stochastic Volatility is another class of models designed to sketch and forecast volatility in data. It can be retraced to theoretical financial literature, formulated by stochastic differential equations in a continuous-time fashion (Heston, 1993). In practice, a discrete-time empirical versions of the SV model are used to address volatility problems as a approximate, especially for time series data (Luo, et al., 2018). The original form of SV model is

$$x_t = \phi x_{t-1} + w_t$$
$$r_t = \beta \exp(\frac{x_t}{2}) v_t$$

Squaring the second equation and taking its logarithm form results in the linear equation,

$$y_t = \alpha + x_t + z_t$$

which formulate the original linear state-space version of the SV model. Notably, the volatility in SV model is no longer conditional deterministic - it is specified as stochastic latent.

In equations above, $\{r_t\}_{t\geq 0}$ is the log-returns on day t, y_t is the logarithm of log-return, and x_t is the logarithm of volatility. When apply SV model to sketch volatility, we use observations (y_t) as input and extract the estimated latent state (x_t). $w_t \sim N(0, r)$, $x_0 \sim N(\mu_0, \sigma_0^2)$, $z_t \sim N(0, 1)$ in a standard SV model.

Hyper Parameters Hyper parameters are set as following: initial state mean (μ_0) = 0, initial state variance (σ_0^2) = 1, observation offset (α) = 1, transition coefficient (ϕ), transition variance (r), observation coefficient and observation variance are all set to 1. ϕ, r, α, μ_0 and σ_0^2 are updated in EM. Iteration time is 100.

Notably, to predict the next volatility x_{t+1} based on the past and current observations $y_{1:t}$, we first calculate the filtered latent means, then predict x_{t+1} based on filtered mean in time t. It works because filtering is forward information delivery, each filtered mean contains information only from past and current observation .

Algorithm Standard SV Model

Fit

```
1 set hyper-parameters as above
2 loop: for iteration i in 0:100
3         E-step: Kalman filtering and Kalman Smoothing
4         M-step: update $\phi$, $r$, $\alpha$, $\mu_0$ and $\sigma_0^2$.
5 obtain the fitted model.
```

Prediction and evaluation

```
6 with fitted hyper-parameters, do Kalman filtering to obtain filtered mean for each time step t
7 for each time step t, predicted latent state $x_t^{pred} = \alpha x_{t-1}^{filtered}$. ($x_0^{filtered} = \mu_0$)
8 calculate rmse bewteen proxy volatility and predicted latent state.
```

The existing package pykalman is used here.

Extension: SV-t model Real return data may have heavier tails which cannot be captured by the standard SV model. We can extend the original SV model to a SV-t model by assuming the observation noise, z_t, follows a student-t distribution. While the emission function is no longer Gaussian, the latent function keeps linear Gaussian.

$$x_t = \phi x_{t-1} + w_t$$

$$y_t = \alpha + x_t + z_t$$

In equations above, $w_t \sim N(0, r)$, $x_0 \sim N(\mu_0, \sigma_0^2)$, $z_t \sim t(n)$ in a SV-t model. We fix the degree of freedom n as 2 in this project.

Hyper Parameters Hyper parameters are set as following: initial state mean (μ_0) = 0, initial state variance (σ_0^2) = 1, observation offset (α) = 1, transition coefficient (ϕ), transition variance (r), observation coefficient are all set to 1, and degree of freedom is 2. ϕ, r, α, μ_0 and σ_0^2 are updated in EM. Iteration time is 100. Amount of samples in particle filter is 1000.

There is a question whether to use smoother or not in EM algorithm. Traditionally Maximization step following particle filtering is based on filtered means and variance. But the gamma message updates are about consistency of uncertainty along the Markov chain. When smoothing the latent mean and variance, there is no need to pass through the observation model again. The derivation only needs transition model to be linear Gaussian, so Kalman smoothing can be applied to SV-t model theoretically. We tried both algorithms on artificial sampled data to determine which one to deploy.

Algorithm SV-t Model without smoothing

```
Fit
    1 set hyper-parameters as above
    2 loop: for iteration i in 0:100
    3     E-step: Particle filtering
    4     M-step: update $\phi$, $r$, $\alpha$, $\mu_0$ and $\sigma_0^2$ with filtered means and variances.
    5 obtain the fitted model.
Prediction and evaluation
    6 Prediction and evaluation steps are the same to a standard SV model, except doing particle
      filtering instead of Kalman filtering.
```

Algorithm SV-t Model with smoothing

```
Fit
    1 set hyper-parameters as above
    2 loop: for iteration i in 0:100
    3     E-step: Particle filtering and Kalman Smoothing
    4     M-step: update $\phi$, $r$, $\alpha$, $\mu_0$ and $\sigma_0^2$ with smoothed means and variances.
    5 obtain the fitted model.
Prediction and evaluation
    6 Prediction and evaluation steps are the same to a standard SV model, except doing particle
      filtering instead of Kalman filtering.
```

Source code from pykalman and the lab code of Particle Filter were cited as reference. Parts about t-distribution were implemented at hand.

2.2.4 Long short-term memory (LSTM) model

LSTM model is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It is powerful time-series model to deal with sequence of data and can predict an arbitrary number of steps into the future.
A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The input gate, decides how much information from current input flows to the cell state; The forget gate, decides how much information from the current input and the previous cell state flows into the current cell state; the output gate, decides how much information from the current cell state flows into the hidden state, so that if needed LSTM can only pick the long-term memories or short-term memories and long-term memories. Based on the design of LSTM, it can deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs.
Here we used previous 10 steps of volatility to next step. We also add three dense layers with activation functions are tanh, relu, and relu respectively. The pseudocode is below:

Algorithm LSTM Model

```
    1 Calculated the proxy volatility.
    2 Normalized the dataset:
      scaler=MinMaxScaler()
      scaler.fit_transform(volatility)
    3 Created the dataset. Use previous 10 steps to predict next step:
      look_back=10
      for i in range(len(volatility)-look_back):
```

```
a = volatility[i:(i+look_back)]
dataX.append(a)
dataY.append(volatility[i+look_back])
return dataX,dataY
```

4 Split the dataset to train and test data:
```
trainX, testX = dataX[0:0.8*len((volatility)], dataX[0.8*len((volatility):len((volatility)]
trainY, testY = dataX[0:0.8*len((volatility)], dataY[0.8*len((volatility):len((volatility)]
```

5 Performed the LSTM classifer. Add three dense layers and fit the model with epochs=30:
```
model = Sequential()
model.add(LSTM(activation='tanh'))
model.add(LSTM(activation='relu'))
model.add(LSTM(activation='relu'))
model.fit(trainX, trainY, epochs=30)
```

6 Conducted prediction:
```
testPredict = model.predict(testX)
```

Library Keras was used in model construction.

3 Experimental Evaluation

3.1 Data

We used the close prices of the following 6 stocks from Dow Jones: Apple (AAPL), Coca-Cola (KO), Goldman Sachs (GS), Home Depot (HD), Johnson Johnson (JNJ) and Walmart (WMT). The date range for the 6 stocks is 2014/06/12-2019/06/11. The actual data we used for log return in our models were:

$$log\,return_t = 100\log(\frac{close\,price_t}{close\,price_{t-1}})$$

We magnified log return 100 times for correct scaling, otherwise the numbers would be too small for most of the algorithms we used.

3.2 Methodology

Data-set Splitting For each of the above data, we chose the integer of $0.8 * lengthofdata$ as the train size, resulting approximately 4/5 of the data starting from the beginning as the train set and the rest 1/5 of the data as the test set.

Evaluation Metrics Since the real volatility is not observable, we used a proxy for the real volatility: $proxy\ for\ volatility\,squared = (log\,return - \overline{log\,return})^2$. Also, since in SV model we transformed the non-linear emission function into linear form by taking logarithm in Stochastic Volatility models, the estimated latent state of SV is the log volatility. Therefore, we used the log of the above proxy in all models for fair evaluation.

$$proxy\ unified = \log(log\,return - \overline{log\,return})^2$$

Finally, we used RMSE to compare this unified proxy for the square volatility of real log return and the estimated and predicted in and out-of sample volatility. We evaluated how well each model performed for the prediction of the volatility based on the values of RMSE.

Validation with Artificial Sampling To check the implement and determine whether to use smoother or not for SV-t model, artificial sampled data was used. It is sampled following a student-t distributed observation noise.

Comparison between models We passed the train data into each of our algorithms and models, and calculated the estimated and predicted values for volatility as described in 3.2 Algorithm. Intuitively and empirically, we expect the models have performance from better to worse in the order of 'LSTM, SV-t, SV, ARIMA-GARCH, GARCH'. This is the hypothesis we aimed to test.

3.3 Results

Figures in this part are presented in section 6.3.

3.3.1 Validation with Artificial Sampling

Results are showed in Figure 2. From standard SV model to SV-t model, it achieves a big improvement in latent state prediction, which consists with our hypothesis that mismatching of assumption in observation model to the actual noise can cause significant estimation error. There is a little enhancement when applying smoother in SV-t model, but the change is not significant, with RMSE from 3.9391 to 3.9388. We decided to use SV-t with smoothing in the following experiments.

3.3.2 Comparison between models

ARIMA, GARCH Figure 3, Figure 4 and Figure 5 show results for ARIMA, GARCH and ARIMA-GARCH respectively. In all figures, proxy was plotted with blue lines, and in-sample prediction and out-of-sample prediction were plotted in yellow and green. The predicted log volatilities by ARIMA model are mostly negative and significantly lower than the real log volatilities, indicating the predicted volatility of ARIMA model are relatively small and ARIMA cannot capture the variance structures very well. The results of GARCH model and ARIMA-GARCH model are much better than the ARIMA results as expected.

Stochastic Volatility models Figure 6 and Figure 7 show the results of simulations from standard SV and SV-t model respectively. In both figures, proxy was plotted with blue lines, and in-sample prediction and out-of-sample prediction were plotted in yellow and green. Compare the two figures, SV-t has achieved a significant progress, sketching the volatility more precisely than what standard SV model does.

LSTM Figure 8 shows the results of simulations from LSTM. The modeled volatility of six stocks (AAPL, KO, GS, HD, JNJ, WMT) were compared with their proxy volatility. Generally, the LSTM can catch the pattern of volatility well since the simulated and proxy volatilities show the similar trend.

Comparison We also reported the RMSEs of LSTM for these six stocks. Table 1 and Table 2 show the RMSEs for training set and testing set respectively. According to the tables, for all the six stocks LSTM model has the best performance, and SV models are significantly better than GARCH models.

3.4 Discussion

Our hypothesis about the performances is generally supported by the experiment results. For each single stock, LSTM performs the best, and SV models beat GARCH models. The results can be explained in terms of the underlying properties of the models. The LSTM model predicts next volatility based on historical volatility. It is straightforward and generative, without extra assumptions about the distributions and lag order. Unlike deep learning models, GARCH and SV model families are developed based on strong assumptions about the relationship between volatility and log-return sequences. GARCH family has even stronger assumptions than SV family. GARCH consider volatility to be conditional on historical residuals and historical volatility, while SV model consider it in a more stochastic way, and take the current information into account. As Chan J.C. and Grant A.L., 2016 demonstrated, the stochastic volatility models generally compare favorably to their GARCH counterparts, indicating that the time-varying volatility is better modeled as a latent stochastic process, especially for financial returns. In our experiment, even a SV with normal distribution assumption performed better than a flexible GARCH with student-t distribution assumption, which further proved it enhanced a lot to specify volatility as stochastic latent.

Table 1: Results for In-sample data (training set)

Stock Name	ARIMA	GARCH	ARIMA-GARCH	SV	SV-t	LSTM
AAPL	5.02	2.93	3.00	2.56	2.46	1.22
KO	4.89	2.94	2.86	3.03	2.74	1.16
GS	4.90	2.72	2.80	2.44	2.39	1.16
HD	5.74	2.71	2.57	2.61	2.37	1.14
JNJ	4.81	2.93	3.00	2.81	2.32	1.16
WMT	4.81	3.00	2.99	2.94	2.64	1.28

Table 2: Results for Out-of-sample data (testing set)

Stock Name	ARIMA	GARCH	ARIMA-GARCH	SV	SV-t	LSTM
AAPL	6.32	3.05	3.15	2.63	2.59	1.29
KO	4.72	2.70	2.66	2.46	2.36	1.10
GS	5.89	2.98	2.66	2.65	2.64	1.24
HD	5.82	2.92	2.94	2.75	2.54	1.20
JNJ	4.75	3.05	3.15	3.01	2.68	1.31
WMT	4.69	2.85	3.05	2.65	2.43	1.15

However, there are two details inconsistent with our expectation. First, it is hard to tell whether GARCH or ARIMA-GARCH is better than the other one. The results of ARIMA-GARCH didn't seem to improve from the GARCH model and indicated that the full ARIMA model didn't help much as a mean model in predicting the volatility of the log return, which in turn made us interested in further exploration in the mean models in the future. Second, for some stocks SV-t model improves not a lot from standard SV model. Since we assume observation noise follows a student-t distribution with degree of freedom of 2, this may result from the inconsistency among the noise for different stocks. To address this problem, degree of freedom should be adjusted in EM algorithm.

4 Conclusion and Future Research

As is known to all, it is hard to model the volatility since there is so much noise in financial market but we predict the mean. Usually models will lead to poor volatility forecasts. In spite of this, we still want to try different models to capture the pattern as possible and see the model improvement.

In conclusion, LSTM has the best capability to capture the pattern of volatility in stock data, and SV-t model performs better than GARCH model. It consists with our expectation that a more generative model would has better performance.Our comparisons among different models could provide suggestions for model choice in volatility prediction in the future.

GARCH and SV models we tried are classic and popular in the application to financial volatility analysis. We should admit that the assumptions in these models are not flexible enough to adapt to variant market environment or to different financial asset. To address this problem, future work may relate to state-switched volatility models, such as HMM-GARCH and HMM-SV. Since there are quantities of extension in the family of volatility model, their application and the approach to implement also deserve further discussion.

We focused on the volatility for uni-dimension, because the most important application of volatility is option pricing, which is specified to single underlying stock. But there could exist dependency among volatilities of different stocks. Multivariant volatility models should be take into consideration, and with which it can process hundreds of stocks together and even probably introduce macro market data into models.

Moreover, as the development of deep learning, more generative models such as RNN perform well in prediction for time series. A hybrid model combining neural network and classic volatility model may be more likely to achieve a breakthrough.

5 Student contributions

Shuang Gao: Data processing; SV and SV-t models implement and analysis; Report write up: Algorithm and Results for SV and SV-t, Methodology, Results, Discussion and Conclusion.

Guojin Tang: Find data source; ARIMA, GARCH and ARIMA-GARCH models analysis; Report write up: Task, Algorithm and Results for ARIMA, GARCH and ARIMA-GARCH, Data and Methodology.

Junrong Zha: Data processing; LSTM model analysis; Report write up: abstract, introduction, related work, Algorithm and Results for LSTM, references.

6 References, Citations, Figures

6.1 References

[1] Bollerslev, T., Engle, R.F. and Nelson, D.B., 1995, Arch Models. Handbook of Econometrics, Eds: Engle, RF and DL Mc Fadden, 4, p.9.

[2] Chan, J.C. and Grant, A.L., 2016. Modeling energy price dynamics: GARCH versus stochastic volatility. Energy Economics, 54, pp.182-189.

[3] Heston, S.L., 1993. A closed-form solution for options with stochastic volatility with applications to bond and currency options. The review of financial studies, 6(2), pp.327-343.

[4] Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.

[5] Luo, R., Zhang, W., Xu, X. and Wang, J., 2018, April. A neural stochastic volatility model. In Thirty-Second AAAI Conference on Artificial Intelligence.

[6]Nkemnole, E.B., Abass, O. and Wulu, J.T., 2017. Forecasting Volatility of Stock Indices with HMM-SV Models.

[7]Panait, I. and Slavescu, E.O., 2012. Using GARCH-IN-mean model to investigate volatility and persistence at different frequencies for Bucharest Stock Exchange during 1997-2012. Theoretical Applied Economics, 19(5).

[8]Taylor, S.J., 1982. Financial returns modelled by the product of two stochastic processes-a study of the daily sugar prices 1961-75. Time series analysis: theory and practice, 1, pp.203-226.

[9]Taylor, S.J., 1986. Modelling financial time series. world scientific.

[10]Zhuang, X.F. and Chan, L.W., 2004, August. Volatility forecasts in financial time series with HMM-GARCH models. In International Conference on Intelligent Data Engineering and Automated Learning (pp. 807-812). Springer, Berlin, Heidelberg.

6.2 Citations

[1]`https://arch.readthedocs.io/en/latest/univariate/univariate_volatility_forecasting.html`

[2]`https://arch.readthedocs.io/en/latest/univariate/introduction.html`

[3]`https://github.com/pykalman/pykalman/blob/master/pykalman/standard.py`

[4]`https://github.com/savinteachingorg/pTSAfall2019/blob/master/labs/week%205/lab-week5-answer.ipynb`

6.3 Figures

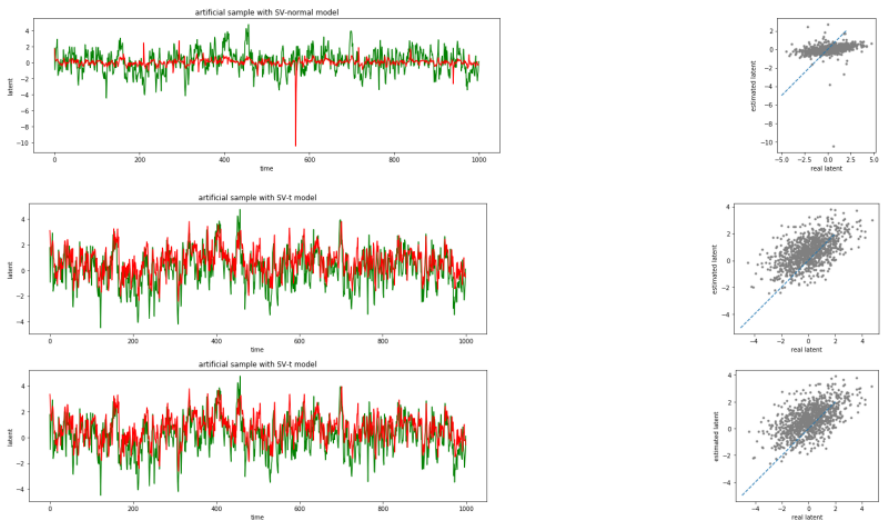

Figure 2: Simulation with artificial sampled data. Standard SV, SV-t without smoothing, SV-t with smoothing are presented orderly.

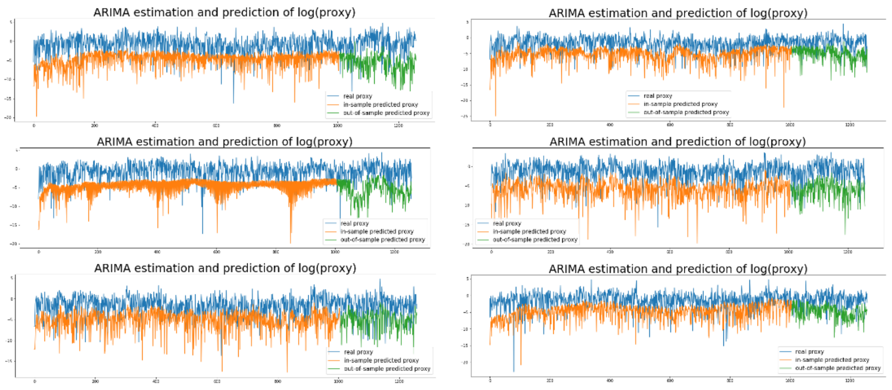

Figure 3: simulated volatility from ARIMA model for six companies: Apple (AAPL), Coca-Cola (KO), Goldman Sachs (GS), Home Depot (HD), Johnson & Johnson (JNJ), Walmart (WMT), from left to right, top to bottom.

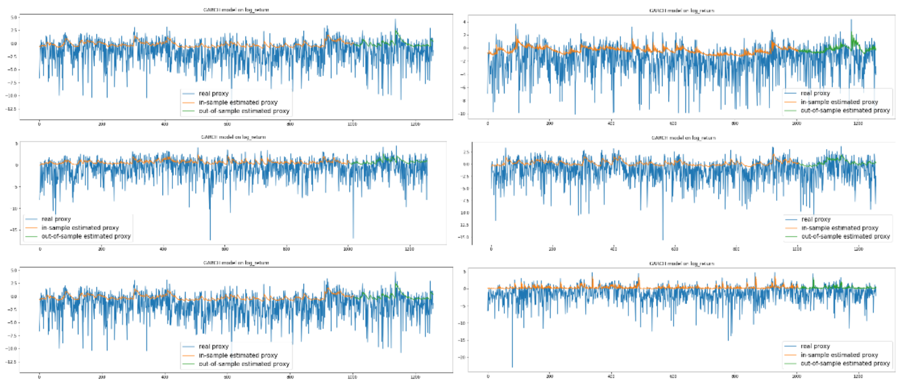

Figure 4: simulated volatility from GARCH model for six companies: Apple (AAPL), Coca-Cola (KO), Goldman Sachs (GS), Home Depot (HD), Johnson & Johnson (JNJ), Walmart (WMT), from left to right, top to bottom.

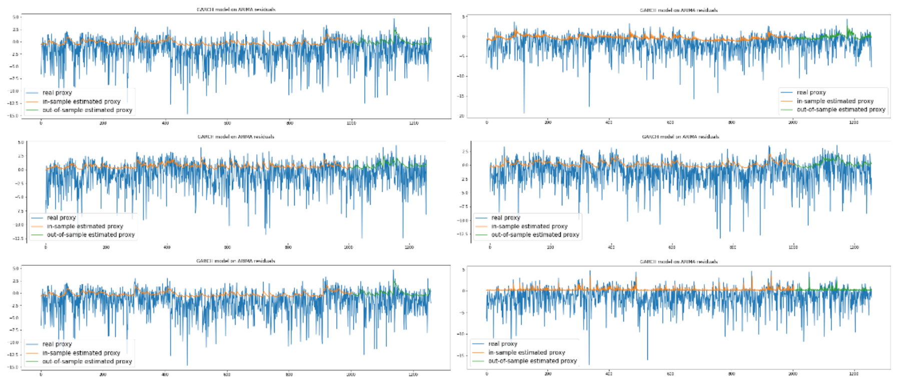
Figure 5: simulated volatility from ARIMA-GARCH model for six companies: Apple (AAPL), Coca-Cola (KO), Goldman Sachs (GS), Home Depot (HD), Johnson & Johnson (JNJ), Walmart (WMT).

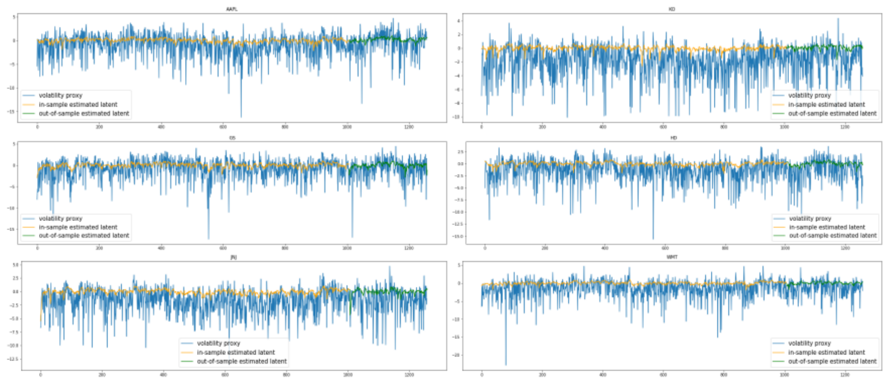
Figure 6: simulated volatility from standard SV model for six companies: Apple (AAPL), Coca-Cola (KO), Goldman Sachs (GS), Home Depot (HD), Johnson & Johnson (JNJ), Walmart (WMT).

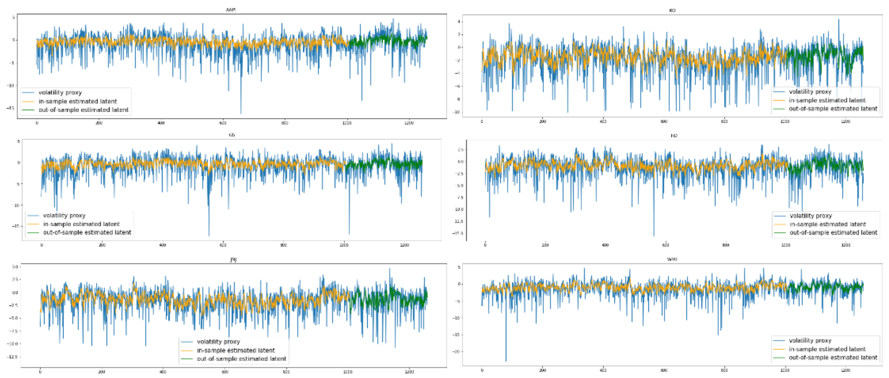
Figure 7: simulated volatility from standard SV-t model for six companies: Apple (AAPL), Coca-Cola (KO), Goldman Sachs (GS), Home Depot (HD), Johnson & Johnson (JNJ), Walmart (WMT).

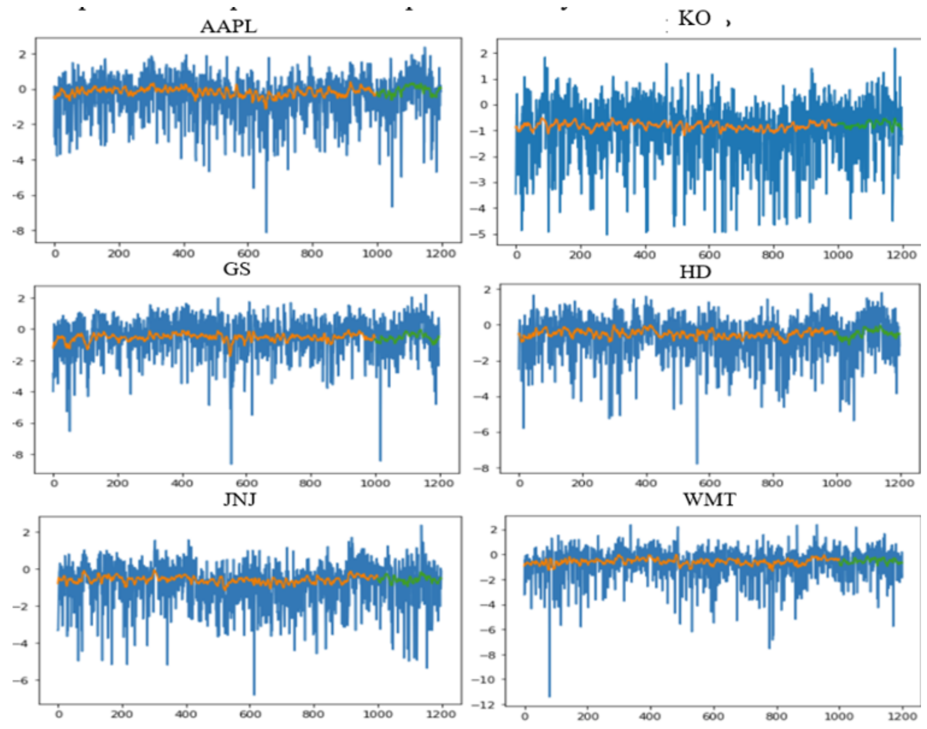

Figure 8: simulated volatility from LSTM for six companies: Apple (AAPL), Coca-Cola (KO), Goldman Sachs (GS), Home Depot (HD), Johnson & Johnson (JNJ), Walmart (WMT).