Machine Learning for Robust Volatility Forecasting

Haider Suleman (hs605), Julian Rovee (jrovee), Sebastian Kronmueller (sekron)

1 Introduction

In financial markets, the return of a financial instrument from time t to $t+1$ is the percentage change in its price between time t and $t+1$, and the volatility of a financial instrument is a measure of the dispersion of its returns. A basic model for an instrument's de-meaned return is $r_t = \sigma_t \eta_t$ where η_t is a random process with mean zero and variance 1, and σ_t is the instantaneous volatility (note that we refer to σ_t as the volatility and its square, σ_t^2, as the variance). Empirically, short term returns tend to be unpredictable. However, the volatility of those returns tends to cluster in ways that make it predictable by relatively straightforward statistical learning techniques.

Being able to predict the future volatility of a financial instrument is important for a variety of reasons, including as an input to portfolio optimization or derivatives pricing.

Our project goal is to forecast the volatility of financial instruments including stocks and cryptocurrencies. The input to each of our models is a timeseries of realized returns. We then use a few different models – classical GARCH models like GARCH-t and beta-t-GARCH, a hybrid SVR-GARCH classical machine learning model, and a deep learning LSTM model – to output the predicted volatility for the next day. Finally, we compare this predicted volatility to our best estimate of the true volatility on that day. In this project, we will be paying special attention to the robustness of the performance of our models across high and low volatility regimes and temporary volatility spikes.

2 Related Work

The problem of predicting volatility involves two steps: estimation and modelling. The main tenet of the estimation literature has been to maximise the of use high-frequency data to obtain low variance estimators of realised volatility that are robust to potential biases such as "microstructure noise" or "jumps". A seminal example is the estimator based on the "subsampling" of Zhang et al. 2005, which averages several realized volatility series computed from intraday data. Pigorsch et al. 2012 provide a survey of several other high-frequency, robust estimators. We will use the method described in former paper to perform our own estimation step.

For the modelling step, Harvey and Chakravarty 2008, Peng et al. 2017, and Jia and Yang 2021, each describe different models for volatility prediction. Harvey and Chakravarty 2008 describe one of the state-of-the-art classical GARCH models (known as Beta-t-GARCH). Peng et al. 2017 describe a hybrid SVR-GARCH model, and Jia and Yang 2021 describe an LSTM approach. The strengths/weaknesses of each of these models will be described in section 4.

One limitation of the approaches in Harvey and Chakravarty 2008 and Jia and Yang 2021 is that these papers use only daily returns as inputs (whereas Peng et al. 2017 uses intraday data). For our machine learning models, we will use intraday data as input, which helps to achieve a higher prediction accuracy.

Additionally, while there is an extensive literature comparing the forecasting performance of various model specifications, most studies tend to confine themselves to one security and one time period. Comparatively few studies evaluate the robustness of their model performance to high-/low volatility regimes, transient spikes in volatility across model types and different asset classes. Another novelty of our approach will be a cross-model comparison of robustness between classical GARCH, non-deep learning Machine Learning, and deep learning models.

3 Dataset and Features

3.1 Data

The Bitcoin data we are using in this study is from a Kaggle dataset (Zielinski 2021) which has trade price data for each minute from December 31, 2011 to March 30, 2021 compiled from various exchange API's. We use volume weighted price data column in order to calculate 5-minute intraday log returns, and then we use these log returns to calculate a realized volatility metric (as described in section 3.3) for each day in our dataset. The equities data used in this paper was extracted from the NYSE Trade and Quote (TAQ 2021). The dataset contains consolidated trades (transaction prices) for 3 liquid stocks in the S&P 500, with symbols IBM, MMM and TGT. The sample period is from 1 January 2002 to 19 November 2021, which gives a total of 5008 trading days. The observations have timestamps accurate to the millisecond frequency. However, for consistency with the bitcoin data, the resolution was reduced to the 1 minute frequency. We further clean the TAQ data following a procedure based on (Brownlees and Gallo 2006), including deletion of non-applicable data and duplications.

3.2 Realized Volatility

Before setting up any models for volatility, we must have a measure for it. Measures of volatility computed from historical returns are known as "realized volatility". This measurement problem is distinct from (and usually precedes) any modelling problems we may consider. As such, realized volatility provides a natural benchmark for forecast evaluation.

Intuitively, estimators for realized volatility begin by assuming that the "true" price process follows a continuous-time Brownian motion, which we only observe, with noise, at discrete time points. Then, the simplest estimator of the realized variance (realized volatility squared) over a day is the square of the return over that day (Andersen et al. 2003). The daily realized volatility of IBM computed using this estimator is plotted in Figure 1. This measure of realized volatility is noisy, as daily returns contain a large idiosyncratic component unrelated to the underlying volatility of the true price process.

We can reduce the *variance* of our realized volatility estimator by sampling the intraday data available to us. However, sampling too frequently would *bias* our estimator as we would begin to pick up "microstructure noise", eg "bid-ask bounce". Empirically, 5-minutes has been shown to be a good sampling frequency (Pigorsch et al. 2012). In this case, the realized variance for a day would be computed as the sum of squared 5-minute returns. Furthermore, to make use of all our data (since we have prices every 1-minute), we also used the "subsampling" scheme of Zhang et al. 2005, whereby we compute the average of five realized volatility series by moving the start time of our 5-minute window by 1-minute increments (also shown in Figure 1).

In the sequel, we address the problems of modelling and forecasting volatility. The GARCH models below attempt to infer and forecast the volatility by using just daily returns, meanwhile the SVR and LSTM models use the realized volatility directly.

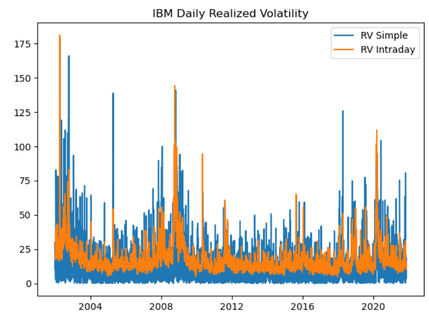

Figure 1: "RV Simple" uses daily and "RV Intraday" uses intraday returns

3.3 Training/Validation/Test Split

In order to evaluate model performance, we set aside 3 years of data (2019, 2020 and 2021) for each financial instrument, to be used as test data. That is, for Bitcoin, our models were trained only on data from 2012 through 2018, and for the equities, our models were trained on data from 2002 through 2018. Importantly, this means that for each instrument, there were high-volatility regimes present in both the training and test data: for equities, both 2008 and 2020 were high-volatility regimes, and for Bitcoin, 2013-2014, 2017-18 and 2020-2021 were high-volatility. This will allow us to evaluate the robustness of our models to different regimes.

Furthermore, both the SVR-GARCH and LSTM models (described below) require some training-validation split for hyperparameter tuning. The schema used for this split will be described later for each model, but in both cases the "validation" data was taken entirely from the training set. Thus, the test data was used only for model evaluation.

Finally, note that we trained the models independently for each financial instrument.

4 Methods

4.1 GARCH and its Descendants

The generalized autoregressive conditional heteroscedastic (GARCH) class of models, introduced by (Bollerslev 1986), are probably the most widely used models for forecasting changing volatility. GARCH models are simple models that use only realized returns as input. Therefore, they will be used as a benchmark for later machine learning models.

Letting Ω_{t-1} denote the information set containing all information up to time $t-1$, the definition of the GARCH class of models begins with the decomposition of returns, r_t, into that one-step ahead conditional variance $\sigma^2_{t|t-1} \triangleq \mathrm{Var}[r_t|\Omega_{t-1}]$ and a random component η_t:

$$r_t = \sigma_{t|t-1}\eta_t \quad \eta_t \sim i.i.d \quad \mathbb{E}(\eta_t) = 0 \quad \mathrm{Var}(\eta_t) = 1 \qquad (1)$$

The GARCH(1,1) model for variance is then defined by the time series model:

$$\sigma^2_{t|t-1} = \omega + \alpha\sigma^2_{t-1|t-2} + \beta r^2_{t-1} \qquad (2)$$

where we constrain the parameters to be non-negative ($\omega > 0,\ \alpha \geq 0,\ \beta \geq 0$) in order to ensure that the variance is always positive and $\alpha + \beta < 1$ to define a strictly stationary process.

While (Bollerslev 1986) initially proposed $\eta_t \sim \mathcal{N}(0,1)$, subsequent works have showed that $\eta_t \sim t_\nu$ (t-distribution with ν degrees of freedom) more satisfactorily captures the fat-tails exhibited by financial returns. Thus, we also adopt this specification. The resulting model is called the GARCH-t.

Training of the model entails learning the unknown parameters $\theta = (\nu, \omega, \alpha, \beta)$. This can be done by maximising the log-likelihood:

$$\ell(\theta) = \sum_{t=1}^{T} \log\left\{\Gamma\left(\frac{\nu+1}{2}\right)\Gamma\left(\frac{\nu}{2}\right)^{-1}\left((\nu-2)\sigma^2_{t|t-1}\right)^{-1/2} \times \left(1 + (\nu-2)^{-1}\sigma^{-2}_{t|t-1}r^2_t\right)^{-(\nu+1)/2}\right\}$$

with respect to the parameter vector θ, subject to the constraints $\omega > 0,\ \alpha \geq 0,\ \beta \geq 0, \nu > 2$ and $\alpha + \beta < 1$. The log-likelihood is highly non-linear in the parameters, therefore a numerical optimiser must be used. Furthermore, the conditional variance $\sigma^2_{t|t-1}$ must also be initialised. If the model is stationary (i.e. if $\alpha+\beta < 1$ is satisfied), then $\sigma^2_{t|t-1}$ can be initialised at its (training) sample counterpart.

A common critique of the GARCH-t model is that the specification of $\sigma^2_{t|t-1}$ as being linearly dependent on past squared return innovations, as in (2), means that it responds too much to extreme observations and the effect is slow to dissipate. In fact, Harvey and Chakravarty 2008 show that while the t-distribution is employed in the predictive distribution of returns and used as the basis for maximum likelihood estimation of the parameters, it is not properly reflected in the dynamic equation (2), which fundamentally still comes from a Gaussian. They suggest updating the dynamic equation for $\sigma^2_{t|t-1}$ to:

$$\sigma^2_{t|t-1} = \omega + \alpha\sigma^2_{t-1|t-2} + \beta\sigma^2_{t-1|t-2}u_{t-1} \qquad (3)$$

where

$$u_t = (\nu+1)r^2_t((\nu-2)\sigma^2_{t|t-1} + r^2_t)^{-1} - 1 \qquad (4)$$

with the same constraints on the parameters as before. Equations (3) and (4) make up the Beta-t-GARCH(1,1) model. The result of the functional form in (4) is a tapering effect for extreme values of r^2_t. Training is as before.

4.2 SVR-GARCH

4.2.1 Support Vector Regression (SVR)

Here, we provide a time-series centric exposition of the of the SVR (Vapnik 1995). Suppose we have T training examples $(\boldsymbol{x}_t, y_t)$ where $\boldsymbol{x}_t \in \mathbb{R}^k$ and $y_t \in \mathbb{R}$, and we have a feature mapping $\Phi : \mathbb{R}^k \to V$ where V is some inner product space. Then we could have a hypothesis:

$$h_t(\boldsymbol{x}_{t-1}) = w_0 + \langle \boldsymbol{w}, \Phi(\boldsymbol{x}_{t-1})\rangle \tag{5}$$

where the parameters are $\boldsymbol{w} \in V$, $w_0 \in \mathbb{R}$, and $\langle \cdot, \cdot \rangle$ denotes the inner product in V.

The SVR adopts an ε-insensitive loss function, which penalizes only the predictions that are greater than some $\varepsilon > 0$ (chosen a priori) away from the true value:

$$L_\varepsilon(y_t - h_t(\boldsymbol{x}_{t-1})) = \max\{0, |y_t - h_t(\boldsymbol{x}_{t-1}))| - \varepsilon\} \tag{6}$$

To estimate parameters, we minimize the regularized loss:

$$\frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w}\rangle + C \cdot \frac{1}{T}\sum_{t=1}^{T} L_\varepsilon(y_t - h_t(\boldsymbol{x}_{t-1})) \tag{7}$$

where the term $\langle \boldsymbol{w}, \boldsymbol{w}\rangle$ characterizes model complexity and C is a constant determining the trade-off between complexity and training error (chosen via cross-validation).

The robustness properties of the loss function (6) are worth noting. Firstly, for points that are strictly inside or outside the ε-boundary, local movements of the target values y_t do not affect the position of $(\boldsymbol{x}_t, y_t)$ versus the boundary, and thus the regression (because the support vectors are unchanged, see Schölkopf et al. 2000). Secondly, the linear tails of the loss function makes the fitting less sensitive to outliers, relative to a typical squared loss function. In fact, (6) corresponds closely with loss functions used in robust regression in statistics (see Huber 1981).

The formulation of the SVR above is known as the "ε-SVR", as it requires us to have knowledge of a desired accuracy ε in advance. Here, we implement a formulation known as the "ν-SVR" (Schölkopf et al. 2000), which automatically minimizes ε. The latter formulation achieves this by including a term $C\nu\varepsilon$ into the objective (7), with additional choice variable ε and additional hyperparameter $\nu \in [0, 1]$. To be concrete, the (re-parametrised) SVR objective becomes:

$$\begin{array}{ll} \text{Minimize:} & \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w}\rangle + C\left(\nu\varepsilon + \frac{1}{T}\sum_{t=1}^{T}\xi_t\right) \\ \text{Subject to:} & |h_t(\boldsymbol{x}_{t-1}) - y_t| \le \varepsilon + \xi_t, \quad \varepsilon, \xi_t \ge 0 \end{array}$$

where $\boldsymbol{w}, w_0$, ε, and ξ_t are decision variables and ν and C are hyperparameters. Here, the slack variable ξ_t is strictly positive for examples with a training loss greater than ε. The interpretation of ν is that it is an upper bound on the fraction of training points with error greater than ε, and a lower bound on the fraction of support vectors.

The SVR is kernelizable, so our predictions can be computed in terms of some kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x})^T\Phi(\boldsymbol{y})$. Here, we will adopt the RBF kernel:

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\|\boldsymbol{x} - \boldsymbol{y}\|_2^2/\tau\right)$$

which is popular because it allows an infinite dimensional feature mapping, Φ, but only has one adjustable parameter.

4.2.2 SVR-GARCH

The GARCH(1,1) dynamic equation, (2), predicts the next period's variance as a linear function of the previous period's variance and squared returns. The SVR can also be used to estimate this dynamic equation, by setting $\boldsymbol{x}_{t-1} = [r_{t-1}^2, \sigma_{t-1}^2]^T$. However, one nuance is that the previous period variance in GARCH, $\sigma_{t-1|t-2}^2$, is not observable a priori. Therefore, a feasible approach is to use our estimate of realized variance from the previous period, say $\hat{\sigma}_{t-1}^2$, instead. We call this the SVR-GARCH(1,1). Our implementation uses the ν-SVR with a RBF Kernel.

One potential advantage of SVR-GARCH over the traditional GARCH is that, as discussed above, the SVR algorithm already has robustness mechanisms built-in. This reduces the risk of our model being thrown off by any transient spikes in volatility. Furthermore, the SVR with a RBF kernel makes no assumptions about the distribution of daily returns. With fewer implicit assumptions, the SVR-GARCH has more freedom to model volatility as some general function of historical volatility/returns, as opposed to just a linear function.

4.2.3 SVR-GARCH Hyperparameter Tuning

Our ν-SVR-GARCH model has 3 hyperparameters: ν (the allowed error rate), C (the regularization coupling), and τ (the radius of the RBF kernel).

Any cross-validation technique we use to tune these hyperparameters must respect the temporal ordering of our dataset. Therefore, standard techniques, such as K-Fold Cross Validation, which assumes that the samples are independently and identically distributed, will not suffice. Below, we describe an alternative cross validation algorithm.

For each triplet (ν, C, τ) such that $\nu \in \{0, 0.01, 0.25, 0.5, 0.75, 1\}$, $C \in \{0.1, 1, 10, 50, 100, 150, 200\}$, and $\tau \in \{0.001, 0.005, 0.01, ..., 100, 500, 1000\}$: (1) Split the training set into $k = 5$ equal-length folds; (2) Train the model on the first fold of data, and compute and store the RMSE of its predictions on the 2nd fold; (3) Add this second fold to the training set, and retrain the model on this augmented training set. Now predict on the 3rd fold. Compute and store the RMSE of these predictions; (4) Repeat this process until we have calculated a prediction RMSE for every fold (except the first); (5) Compute the mean of the stored RMSE's. This is the "loss" assigned to the particular (ν, C, τ) triplet. Finally, choose the triplet (ν, C, τ) with the lowest average loss. Intuitively, our cross-validation scheme involves an expanding training set with a fixed length validation set.

For each of the equities time-series, the optimal hyperparameters were $(\nu, C, \tau) = (0.5, 10.0, 100)$, while for Bitcoin they were $(\nu, C, \tau) = (0.75, 200.0, 1000)$. Training and hyperparameter selection for SVR-GARCH was implemented using scikit-learn.

4.3 Long Short Term Memory Networks

4.3.1 Background on LSTM Networks

To evaluate the performance of deep learning models to predict volatility, we are using modules based on the Long Short-Term Memory algorithm introduced in Hochreiter and Schmidhuber 1997. This algorithm is a variant of a recurrent neural network architecture, where the output of the network of input t feeds into into the next cell of the network including the next input at t+1, but it also includes a ”cell state” track that gets passed between cells with only linear modifications, addressing the vanishing gradient problem and thus allowing information to be passed between cells with longer distances. The cells itself contain several computations impacting both output and carry state, for details see Goodfellow et al. 2016 p.397ff. We apply LSTM networks to predict volatility as, as with SVR-GARCH, we can use them to directly model the realized volatility, requiring no assumptions relating to the data generating process of the returns, leading to higher model flexibility. Further, we expect the LSTM models to be more robust than the classical GARCH models to temporary volatility spikes. With a lookback window of a certain set of periods, the LSTM has contextual information about the true underlying volatility for each sample that can't be represented by the fixed factors in the GARCH models. For the implementation of the models we use Keras/Tensorflow and take inspiration from the approach outlined in Chollet 2017 p.207ff.

4.3.2 LSTM Model Baseline Architecture

Our baseline model architecture uses a combination of LSTM cells, a dense layer and an output layer with one cell and an input sequence of T days. We also use a dropout between the LSTM layer and the dense layer to prevent overfitting. For a graphical illustration of the baseline architecture, see figure 2.

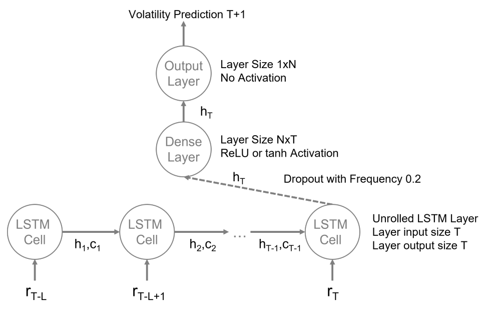

Figure 2: Baseline network architecture

We fix $T = 32$, representing around 6 weeks of trading data. This choice of T allows us to consider complex input patterns leading to increased/decreased volatility while taking into consideration that the correlation of financial data over time fades quickly. As input features for each day we use both the calculated realized volatility and the daily returns, see discussion in section 2. This also allows us to set the realized volatility of the day directly following the 32 input days per sample as training target, instead of the returns as in other approaches in literature, and allows us to use mean square error as loss function (Instead of e.g., the Gaussian loss used in Jia and Yang 2021 needed to predict volatility based on daily returns).

4.3.3 LSTM Hyperparameter Tuning

To find the best hyperparameters for the LSTM model, we follow a two step approach. In the first step, we conduct a grid search varying the number of LSTM cell units (32,64,128), the dropout rates (0,0.2,0.5), the number of dense units (0,40) and the activation function of the dense layer (tanh and ReLu). All models are trained for 250 epochs on a batch size of 256 samples. The data used for hyperparameter search is the IBM stock data from 2002 until 2018, temporally split into 80% for training and 20% for validation. The result of the grid search (see figure 3) shows that models with a higher number LSTM cells (128) and / or no dropout layer tended to overfit. Meanwhile, the activation function, dropout strength, and dense layer seemed to have no systematic effect on model performanece. As a second step, we pick the highest performing model - with 64 LSTM units, 40 dense units, a 0.5 dropout and tanh activation and train it for 1000 epochs to find the ideal training length for the model. Based on the results, we select 250 epochs as ideal training length for the model. See figure 4 for the training / validation loss versus training epochs of the model.

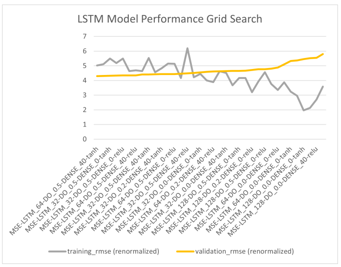

Figure 3: Results of model optimization grid search

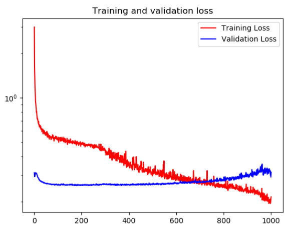

Figure 4: Final model loss versus epochs.

5 Results & Discussion

5.1 Benchmarking Methodology

After fitting the models to the training set, we use them to generate one-step-ahead forecasts in the test set. Two metrics were used to evaluate model performance on the test set: root mean squared error (RMSE) of prediction vs. realized volatility , and mean absolute error (MAE) of prediction vs. realized volatility. RMSE is the most common error metric for evaluating regression models. However, RMSE also penalizes large deviations from the target value more heavily than a linear loss like MAE. For the sake of completeness, we have included both RMSE and MAE in our report. Furthermore, we report the test set performance of our models split by year. This will illuminate whether certain models tend to perform better in high volatility regimes (such as 2020). Finally, the SVR-GARCH optimises a (piecewise) linear cost function while the LSTM optimizes MSE. This is another reason for using both MAE and RMSE.

5.2 General Model Performance

The test set performance of the models is summarized in Table 1. Figure 5 visualizes the relative prediction performance for TGT on both the training and test sets. The graphs for other instruments show similar behavior. Qualitatively, there is little difference in the relative test set performance of the models as judged using RMSE vs MAE. Thus, we restrict our discussion here to the RMSE results, and note that it largely extends to the MAE results. For the three equities (IBM, MMM, TGT), Table 1 shows that both the SVR-GARCH and the LSTM significantly outperformed the classical GARCH models for all years in the test set. The LSTM and SVR-GARCH had a similar overall performance on the test set, with the the SVR-GARCH performing marginally better. It is noteworthy that the LSTM performed better during the low volatility regimes (2019 and 2021), whereas the SVR-GARCH performed better during the high volatility regime (2020). This is somewhat expected, as no high-volatility regimes were included in the validation set for the LSTM. Separately, note that the robustification due to the Beta-t-GARCH specfication led to its superior performance over the GARCH-t. For Bitcoin, the LSTM significantly outperformed all other models on the test set for all years. This is partially explained by the fact that the high-volatility regimes are more dispersed for Bitcoin than for the equities, so the LSTM's performance is not let down by one particular period.

RMSE	**Train**	**Test**			
		All	***2019***	***2020***	***2021***
IBM					
GARCH-t	7.55	9.25	6.20	13.18	5.70
Beta-t-GARCH	7.20	8.97	6.03	12.93	5.07
SVR-GARCH	5.90	7.07	5.69	9.66	4.29
LSTM	4.74	7.18	5.23	10.17	4.11
MMM					
GARCH-t	7.40	9.76	6.09	14.34	5.29
Beta-t-GARCH	7.22	9.28	6.29	13.41	5.10
SVR-GARCH	6.38	7.23	5.99	9.66	4.68
LSTM	5.60	7.56	5.46	10.68	4.44
TGT					
GARCH-t	9.22	12.04	11.34	14.89	8.64
Beta-t-GARCH	9.13	11.35	7.84	15.79	7.67
SVR-GARCH	8.16	9.07	8.14	11.10	7.22
LSTM	6.68	10.53	7.19	14.79	6.84
Bitcoin					
GARCH-t	60.88	29.69	26.52	29.63	39.51
Beta-t-GARCH	61.44	36.25	34.27	37.29	38.96
SVR-GARCH	54.60	29.33	27.45	29.15	36.11
LSTM	46.07	27.57	24.18	28.52	34.64

MAE	**Train**	**Test**			
		All	***2019***	***2020***	***2021***
IBM					
GARCH-t	4.70	5.70	4.27	7.95	4.56
Beta-t-GARCH	4.54	5.35	3.91	7.77	4.05
SVR-GARCH	3.38	4.06	3.56	5.54	2.87
LSTM	2.90	4.14	3.24	6.11	2.79
MMM					
GARCH-t	4.56	6.31	4.84	9.58	4.07
Beta-t-GARCH	4.43	5.88	4.93	8.49	3.87
SVR-GARCH	3.48	4.34	4.09	5.63	3.12
LSTM	2.98	4.32	3.67	6.10	2.95
TGT					
GARCH-t	6.06	7.50	7.06	8.94	6.29
Beta-t-GARCH	5.98	6.27	4.91	8.56	5.04
SVR-GARCH	5.27	4.91	4.55	5.93	4.10
LSTM	4.67	5.31	4.37	7.18	4.13
Bitcoin					
GARCH-t	29.57	18.48	18.08	17.22	25.12
Beta-t-GARCH	32.62	25.66	26.16	23.79	31.53
SVR-GARCH	22.19	16.85	16.59	15.50	23.39
LSTM	19.45	15.28	14.37	14.31	22.73

Table 1: RMSE and MAE vs Realized Volatility

Finally, as expected (cf. Section 5.1), the magnitude of the loss is higher in high volatility regimes (e.g. 2020 for equities) than in low volatility ones, vindicating our decision to also consider MAE. Further, we observe that model performance does not seem to degrade over the test set. To see this, note that the magnitudes of the losses are similar in 2019 and 2021 for the equities and that these two years had a similar level of volatility. The latter fact can be verified by looking at the bottom plot of Figure 5. Thus we expect that using an expanding training set would not affect our conclusions, especially because the 3 years of test data are small relative to the 17 years of training data.

5.3 Robustness

First, we note that the GARCH-t is susceptible to being "thrown-off" by transient spikes in volatility. This can be seen from the bottom plot of Figure 5. That is, at a date close to the end of 2019, there was a sharp spike in volatility of TGT. While all models successfully adapted to the steep rise, the GARCH-t predictions failed to return to the true volatility for several months. The Beta-t-GARCH was also found to be susceptible to such spikes, but to a lesser extent. For example, the predictions of both the classical GARCH models lingered at too high a level after the peak COVID-related volatility spike in early 2020. Holistically, Figure 5 suggests that the superior performance of the Machine Learning models was due to their robustness to such spikes. Tellingly, the robustifcation due to Beta-t-GARCH failed to deliver a superior performance to the GARCH-t for Bitcoin. This shows the difficulty of depending on the researcher to specify a robust parametric form. On the other hand, Machine Learning models provide us with a highly flexible class of non-linear models, which puts less burden on the researcher to specify a desirable functional form. Finally, our findings are robust to using different metrics (RMSE/MAE) and various asset classes (equities and cryptocurrencies). That is, machine learning models were found to be generically superior to classical GARCH models for volatility forecasting.

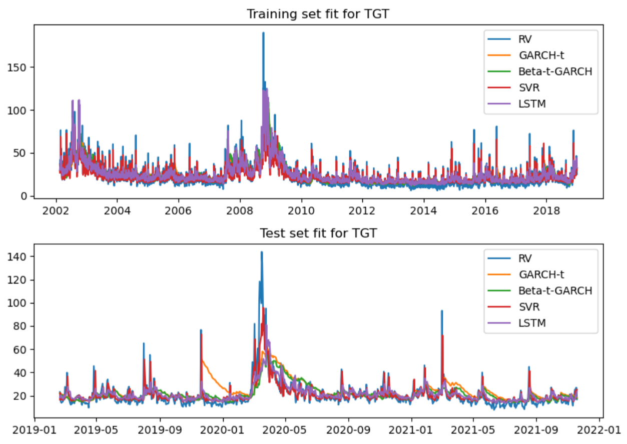

Figure 5: TGT: Training (top) and Test set (bottom) fit

5.4 Conclusion

In conclusion, we successfully implemented and benchmarked several models to predict volatility. We also showed that both the SVR-GARCH and the LSTM model perform better and are more robust to sharp volatility spikes than the classical GARCH models. In practice the final selection of the right model will depend on use case, computational resource availability and the required prediction performance. Further extensions of our work are possible. For example, the inclusion of several different stocks at once as input to the LSTM, enabling the inclusion of "spillover" effects and thus potentially even further improving the model performance.

6 Contributions

All three authors contributed equally to this project. The write up and assessment of the models is joint work of all authors. For the model implementation and optimization, hs605 worked on the classical GARCH models, hs605 and jrovee worked on the SVR-GARCH model and sekron worked on the LSTM model.

References

Andersen, Torben G. et al. (2003). "Modeling and Forecasting Realized Volatility". In: *Econometrica* 71.2, pp. 579–625. ISSN: 00129682, 14680262. URL: `http://www.jstor.org/stable/3082068`.

Bollerslev, Tim (1986). "Generalized autoregressive conditional heteroskedasticity". In: *Journal of Econometrics* 31.3, pp. 307–327. ISSN: 0304-4076. DOI: `https://doi.org/10.1016/0304-4076(86)90063-1`. URL: `https://www.sciencedirect.com/science/article/pii/0304407686900631`.

Brownlees, C. and G. Gallo (2006). "Financial econometric analysis at ultra-high frequency: Data handling concerns". In: *Computational Statistics and Data Analysis* 51.4, pp. 2232–2245.

Chollet, François (Nov. 2017). *Deep Learning with Python.* Manning. ISBN: 9781617294433.

Goodfellow, Ian J. et al. (2016). *Deep Learning.* `http://www.deeplearningbook.org`. Cambridge, MA, USA: MIT Press.

Harvey, Andrew and Tirthankar Chakravarty (2008). "Beta-t-(E)GARCH". In: *Cambridge Working Papers in Economics.* DOI: `https://doi.org/10.17863/CAM.5286`.

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`. eprint: `https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf`. URL: `https://doi.org/10.1162/neco.1997.9.8.1735`.

Huber, P.J. (1981). *Robust statistics.* Wiley New York.

Jia, Fang and Boli Yang (2021). "Forecasting Volatility of Stock Index: Deep Learning Model with Likelihood-Based Loss Function". In: *Complexity* 2021.

Peng, Yaohao et al. (Dec. 2017). "The best of two worlds: Forecasting High Frequency Volatility for cryptocurrencies and traditional currencies with Support Vector Regression". In: *Expert Systems with Applications* 97. DOI: `10.1016/j.eswa.2017.12.004`.

Pigorsch, Christian et al. (2012). "Volatility Estimation Based on High-Frequency Data". In: *Handbook of Computational Finance.* Ed. by Jin-Chuan Duan et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 335–369. ISBN: 978-3-642-17254-0. DOI: `10.1007/978-3-642-17254-0_13`. URL: `https://doi.org/10.1007/978-3-642-17254-0_13`.

Schölkopf, Bernhard et al. (May 2000). "New Support Vector Algorithms". In: *Neural Computation* 12.5, pp. 1207–1245. ISSN: 0899-7667. DOI: `10.1162/089976600300015565`. eprint: `https://direct.mit.edu/neco/article-pdf/12/5/1207/814467/089976600300015565.pdf`. URL: `https://doi.org/10.1162/089976600300015565`.

TAQ (2021). *NYSE Trade and Quote (TAQ).* URL: `https://wrds-www.wharton.upenn.edu/pages/about/data-vendors/nyse-trade-and-quote-taq/`.

Vapnik, Vladimir N. (1995). *The Nature of Statistical Learning Theory.* Berlin, Heidelberg: Springer-Verlag. ISBN: 0387945598.

Zhang, Lan et al. (2005). "A Tale of Two Time Scales: Determining Integrated Volatility with Noisy High-Frequency Data". In: *Journal of the American Statistical Association* 100.472, pp. 1394–1411. ISSN: 01621459. URL: `http://www.jstor.org/stable/27590680`.

Zielinski, Mark (Aug. 2021). *Bitcoin Historical Data.* Version 7. URL: `https://www.kaggle.com/mczielinski/bitcoin-historical-data/version/7`.