Master Thesis
MSc 203 - Financial Markets

# Realized Volatility forecasting using Machine Learning.

Alban Magerand
Tutor: Guillaume Andrieux

May 2023

# Abstract

This master's thesis was written during my internship within the Delta-One trading team at Société Générale as part of my MSc 203 curriculum. The code used to produce the results can be found in this Github repository and any discussion about it is more than welcome.

The objective of this paper is to investigate and build on various models related to realized volatility forecasting, ranging from classic econometrics techniques, such as the Ordinary Least Squares method, to more advanced machine learning approaches, e.g., XGBoost.

The study begins by presenting empirical characteristics of volatility that need to be considered when building a forecasting model, followed by an explanation of the mathematical foundations underlying these models. A universe of stocks from the S&P500 is defined to evaluate the accuracy of the models and assess their reliability. The most prominent features used in forecasting are also identified and analyzed.

Overall, this thesis contributes to the understanding and improvement of realized volatility forecasting, which has crucial implications for financial risk management and trading strategies.

# Contents

# Introduction

*'And I don't dare write down any sort of formal model of the process by which volatilities change. I'm not sure I ever will.'* These are the two closing sentences written by Black (1976) in his Studies of Stock Price Volatility Changes. Clearly, the modelisation and the forecasting of volatility have been central subjects of research in Quantitative Finance for the past decades, and this can be easily understood given its importance in what we call modern finance. Indeed, volatility is one the key inputs in the pricing of a sizeable number of derivatives, most risk measures rely on it such as the V.a.R., and derivatives with volatility itself as the underlying are gaining more and more interest.

The first breakthrough in this field came in 1982 with Engle, shortly after Black's initial reflexion, where he proposed the initial ARCH model. Then in 1986, Bollerslev built the GARCH model from Engle's work, and was still used as a reference for volatility forecasting not so long ago. Stochastic volatility models have also been explored, in particular by Taylor (1986) which model had the capacity to capture the empirical observation of volatility clustering. These models do perform well on long time frames (at least a day), but have been progressively replaced by models capable of dealing with the appearance of high frequency data.

A new page in the history of volatility forecasting was written with Corsi in 2003, who proposed the HAR (Heterogeneous Auto Regressive) model, which specifies realized volatility as a linear function of daily, weekly and monthly returns. This model is economically driven in addition to its excellent forecasting abilities by accurately reflecting the choices of investors with various time horizons: the length of the lags enables to take into account the attitude of the different kinds of participants in the market. Numerous models have been and are still being developed based on this branch of the literature, all trying to capture as much stylized facts about volatility as possible.

While today it still seems that Black was right, the increasing use of Machine Learning in this field of Quantitative Finance has led to promising results. Recent studies (from 2019 until today) have shown that incorporating Artificial Intelligence in the development of new models leads to promising results. Indeed, M.L. techniques offer numerous advantages over traditional econometric models in the realm of volatility forecasting. By leveraging large datasets and sophisticated algorithms, machine learning models are able to capture complex patterns and relationships in market data that may be missed by linear models. Furthermore, traditional econometric models often require strong assumptions to be made about the underlying data, whereas machine learning models are generally assumption-free, allowing them to adapt more easily to non-linear and non-parametric relationships in the data.

In this master's thesis we will focus on the features selection of our model in order to predict the next day realized volatility. To do so, we will first use conventional models which are still widely used and test different ways to estimate the coefficients on a universe of fifteen stocks from the S&P500. Next we will compare their performances with Machine Learning

models (Gaussian Process Regression and XGBoost), which hyperparameters will be tuned using various techniques. Lastly we will present the overall results of this study as well as suggestions to explore in order to broaden this study.

# Chapter 1

# Volatility

## 1.1 Different volatility measures

### 1.1.1 Historical volatility

The volatility of a financial product is a measure of the dispersion of its returns around their mean, and is usually denoted by $\sigma$. It is mathematically defined as:

$$\sigma = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(r_t - \mu)^2} \tag{1.1}$$

where $r_t = ln(\frac{S_t}{S_{t-1)}}$, $S_t$=price of the asset at time t, $\mu$= mean of $r_t$ and $n$=number of observations.

To estimate the population variance from the sample variance, the following unbiased estimator is commonly used:

$$\sigma^2 = \frac{n-1}{n}s^2 \tag{1.2}$$

where $s^2$= variance in the sample.
To obtain an estimator of the volatility, it is not sufficient to take the square root of $s^2$. Indeed, using Jensen's equality, we know that:

$$\mathbb{E}[\sqrt{s^2}] \leq \sqrt{\mathbb{E}[s^2]} = \sigma \tag{1.3}$$

Assuming that the returns are n.i.i.d. with $r_i \sim \mathcal{N}(\mu, \sigma^2)$, we can show using Cochrane's theorem that:

$$\mathbb{E}[s] = \sqrt{\frac{2}{n-1}}\frac{\Gamma(\frac{n}{2})}{\Gamma(\frac{n-1}{2})}\sigma = h(n)\sigma \tag{1.4}$$

where $x \mapsto \Gamma(x)$ is the Gamma function.

As we can see in Figure 1, $n \mapsto h(n)$ tends to 1 rapidly, making therefore the estimator of the volatility close to its "true" value rapidly:

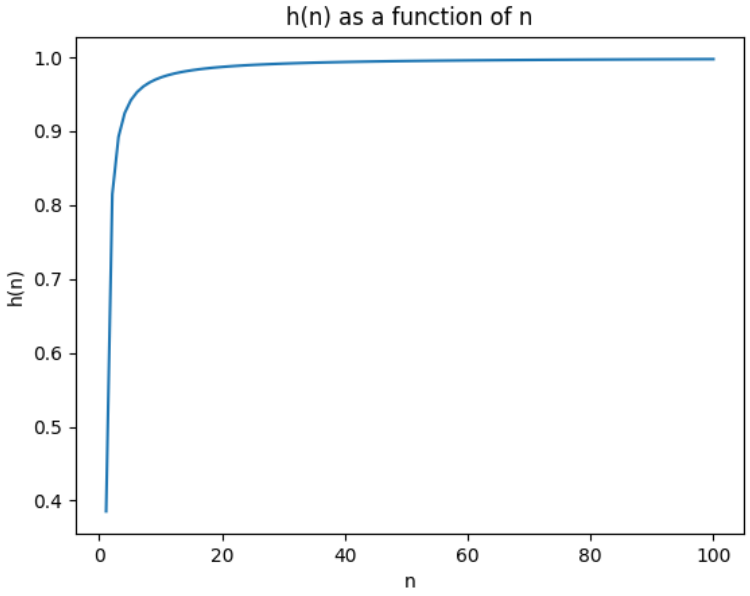


Figure 1.1: Illustration of the speed of convergence of the volatility estimator towards its true value.

The volatility we have been focusing on so far is called historical volatility. As we have shown, it can be computed using first an estimator of the mean of the returns, and then an estimator of the volatility itself. However, when working on small samples of data or during particularly agitated markets conditions, estimations of the mean returns tend to become noisy and hardly interpretable.

### 1.1.2 Realized volatility

Given the flaws identified when using historical volatility, and with the increasing availability of high frequency data, it became the norm among practitioners to set the mean return $\mu$ to 0. This volatility is often estimated on shorter time-frames than the historical one, for example using intraday returns.
Denoting $X(t)$ the logarithmic price process of a security, we assume that $X(t)$ follows a classic Itô process:

$$X_t = \mu(t)dt + \sigma(t)dB(t) \tag{1.5}$$

where $\mu(t)$ = drift term, $\sigma(t)$ = diffusion term and $B(t)$ is a Brownian motion.
We then define the continuously compounded return between $t-\delta$ and $t$, with $0 \leq \delta \leq t$:

$$r(t,\delta) = X(t) - X(t-\delta) = \int_{t-\delta}^{t} \mu(\tau)d\tau + \int_{t-\delta}^{t} \sigma(\tau)dB(\tau) \tag{1.6}$$

as well as its cumulative volatility over a period of time $[t-\delta, t]$, which is then given by:

$$QV(t,\delta) = \int_{t-\delta}^{t} \sigma^2(t)dt \tag{1.7}$$

We are finally equipped to define the realized volatility over a time period $[t-\delta, t]$:

$$RV(t,\delta,n) = \sqrt{\sum_{i=1}^{n\cdot\delta} r^2(t-\delta+\frac{i}{n}, \frac{1}{n})} \tag{1.8}$$

It is important to note that in the classic theory of stochastic processes, we have the following convergence:

$$RV^2(t,\delta,n) \xrightarrow{n\to+\infty} QV(t,\delta) \tag{1.9}$$

This relation does not hold anymore when using high frequency data, where returns are sampled on an extremely short basis. Indeed, it has been observed empirically that the sum of the squared returns, instead of converging toward the cumulative volatility increases due to noises in the market microstructure. In order to avoid these issues, the usage in the current literature is to use at least a 5 minutes timeframe to compute the returns.
If we were to work high frequency data, i.e., with a sampling time inferior to 5 minutes in the case of volatility forecasting, then the work of Christensen and Podolskij (2007) would provide a valuable help with regard to using an appropriate estimator. Even though there does not exist to this day a definition of the most accurate estimator of cumulative volatility, the one provided by the two authors is called realized range-based variance and has been proven to provide a much smaller variance than classic R.V. (up to 2/3 of its variance). It has been defined as:

$$RRV_m^{\Delta} = \frac{\sum_{i=1}^{n} s_m^2}{\lambda_m} \tag{1.10}$$

where $\lambda_m$ is a constant, and $s_m^2$ is the observed range of price for an asset over the interval $[\frac{i_1}{n}, \frac{i}{n}]$ with $i \in [1,n]$ and $m$ being the number of results used to obtain $s_m^2$.

This is one of the few stylized facts about high frequency data, and we will go into more details about the ones of volatility later in this paper.

### 1.1.3 Implied Volatility

Contrarily to historical and realized volatility, implied volatility does not rely on past data to be estimated. It is by essence the estimated standard deviation of the log-return of the underlying up to expiry, which is why it is said to be "forward-looking".

In order to estimate I.V., one should first define an option pricing model, the most commonly used being the Black-Scholes one. Then, from the prices of options observed in the market and using numerical methods to find the I.V. value that fits the model to reality, it is possible to obtain the implied volatility of the underlying.
For example using Black-Scholes model, the price of a call option with strike $K$ and maturity $T$ is given by:

$$c(S_0,T,\sigma) = S_0 N(d_1) - Ke^{-rT}N(d_2) \tag{1.11}$$

with

$$d_1 = \frac{ln\left(\frac{S_0}{K}\right) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \quad \text{and} \quad d_2 = d_1 - \sigma\sqrt{\tau}$$

Even though there exists no closed-end formula to deduce the implied volatility from this model and observed call price on the market, using an iterative scheme such as Newton one enables us to obtain the volatility expected by the market.

## 1.2 Empirical observations

Thanks to the increasing integration of data analysis in trading, patterns have emerged in the behaviour of volatility. This is a crucial point, since a good performing model should try to capture these facts in order to be as close to the market as possible.

### 1.2.1 Volatility clustering

Volatility clustering is a statistical phenomenon observed in financial markets where periods of high volatility tend to be followed by more periods of high volatility, and periods of low volatility tend to be followed by more periods of low volatility. This phenomenon is often observed in the prices of stocks, bonds, and other financial instruments.
This is a well-documented feature of financial markets and is often attributed to the interplay between market participants' emotions and rational decision-making processes. In times of high volatility, investors may become fearful or anxious and make more emotional trading decisions. This can lead to a self-reinforcing cycle of selling that causes prices to fall even further and increases volatility. Conversely, in times of low volatility, investors may become complacent and take on more risk, leading to a self-reinforcing cycle of buying that causes prices to rise and volatility to remain low.
Volatility clustering is important for investors and traders to understand because it can have a significant impact on portfolio performance. In periods of high volatility, portfolio values can experience significant swings, making it difficult to predict future returns. Conversely, in periods of low volatility, portfolio values may not move as much, making it easier to predict future returns, but also potentially reducing overall returns if trading decisions are not adjusted accordingly.
To illustrate this, let us have a look at the S&P500 between the $1^{st}$ January 2000 and the $5^{th}$ of March 2023. We compute the realized volatility over a one month-period and plot it against the realized volatility of the previous month, which formally gives:

$$RV(M+1, D, D) = RV(M, D, D)$$

where $M$ means Monthly and $D$ means Daily. We obtain the following graph:

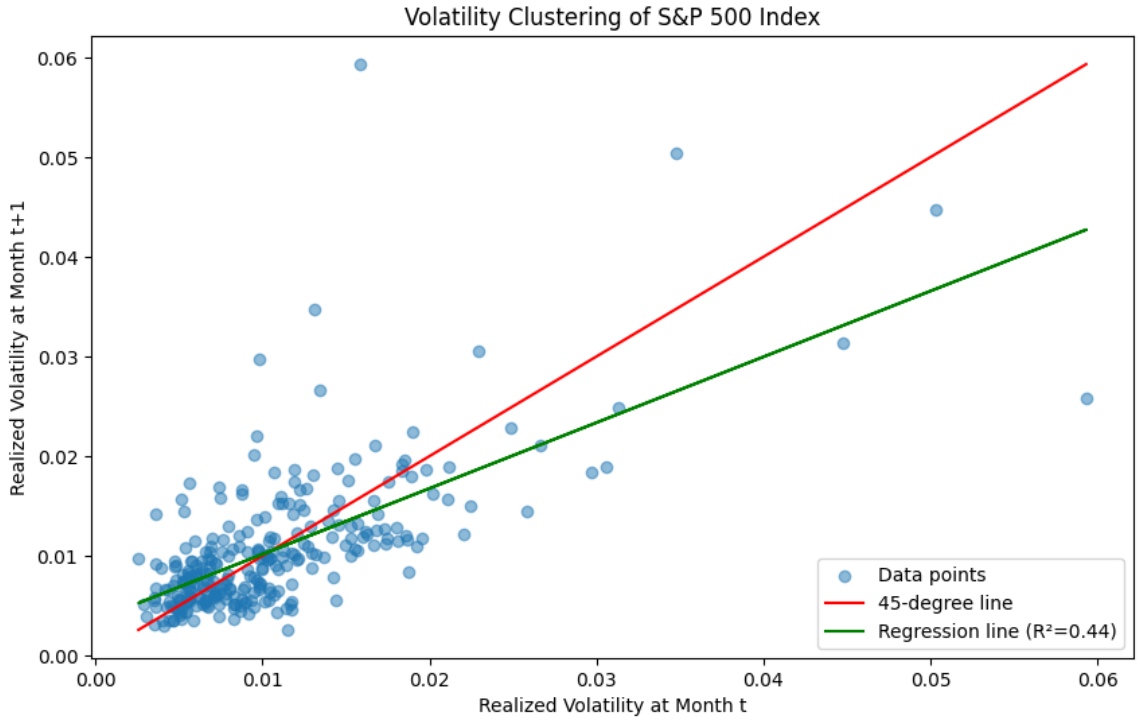


Figure 1.2: Illustration of the volatility clustering effect on the S&P.

The red line being displayed represents the line of equality, where the $RV(M+1) = RV(M)$.

Points above the line indicate that the volatility in t+1 is higher than the volatility in t, while points below the line indicate that the volatility in t+1 is lower than the volatility in t. The greater the deviation from the line of equality, the stronger the volatility clustering effect, as it indicates that periods of high volatility tend to be followed by more periods of high volatility.

As we can see, the linear regression (green line) with the lagged volatility as explanatory variable does not fit particularly well,especially for outliers or days with high volatility. This is a real problem, since it is especially when markets are turbulent that models are important in order to manage risk and make well informed trading decisions. This is due to another specificity of realized volatility, which is its mean-reverting aspect.

### 1.2.2 Mean reversion

The empirical finding of the mean reverting property of volatility suggests that, over time, volatility tends to return to its long-term average or mean. This means that periods of high volatility are likely to be followed by periods of lower volatility, and vice versa. This stylized fact is closely linked to volatility clustering, since aggregating both means that periods of high volatility tend to cluster together because of the underlying market dynamics and shocks that cause volatility to spike. However, these spikes are typically short-lived, and volatility tends to revert to its mean over time. This means that periods of high volatility are often followed by periods of lower volatility, as market participants adjust their expectations and actions to reflect the new information.
To visualize this, we plot in a first time the evolution of realized volatility around its mean:

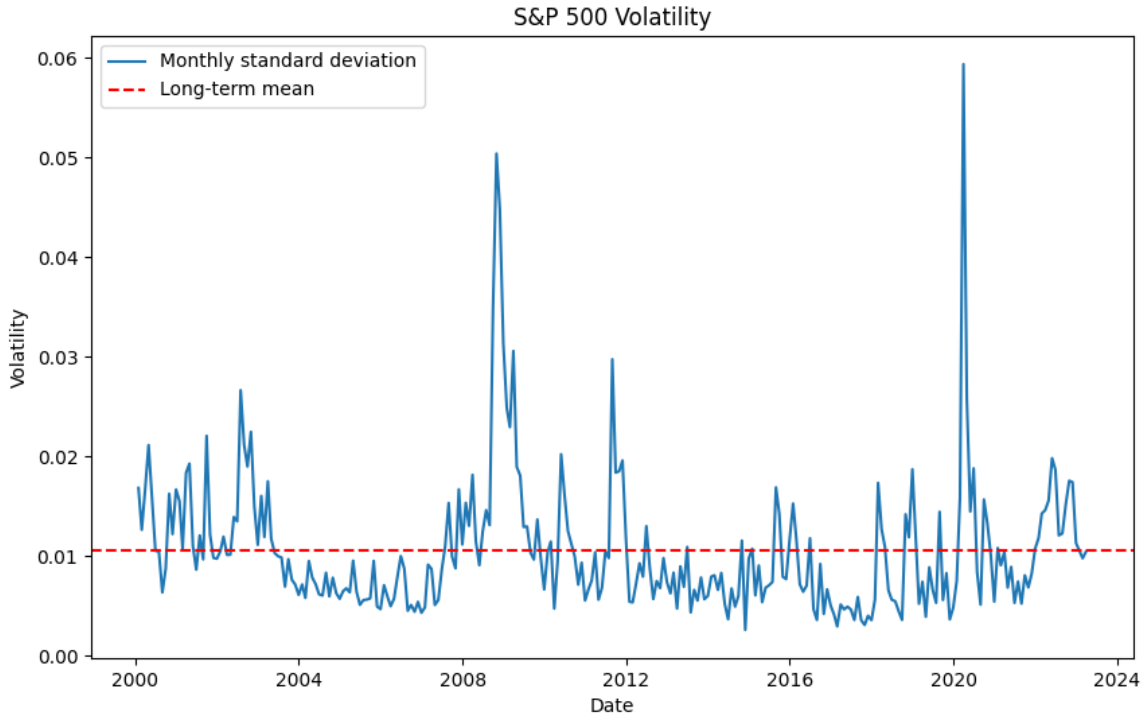


Figure 1.3: Evolution of the monthly volatility of the S&P around its mean.

Despite showing that volatility does stay "close" to its mean and goes back around it after huge spikes, this does not tell us much about the speed of convergence, or if convergence there is at all.

To get a more precise idea of this, we create two baskets containing the highest and lowest 10% of all the observations of monthly realized volatility. Then, we look at the evolution over the next months for each these data sets, and take the mean month by month. This results in the following graph:

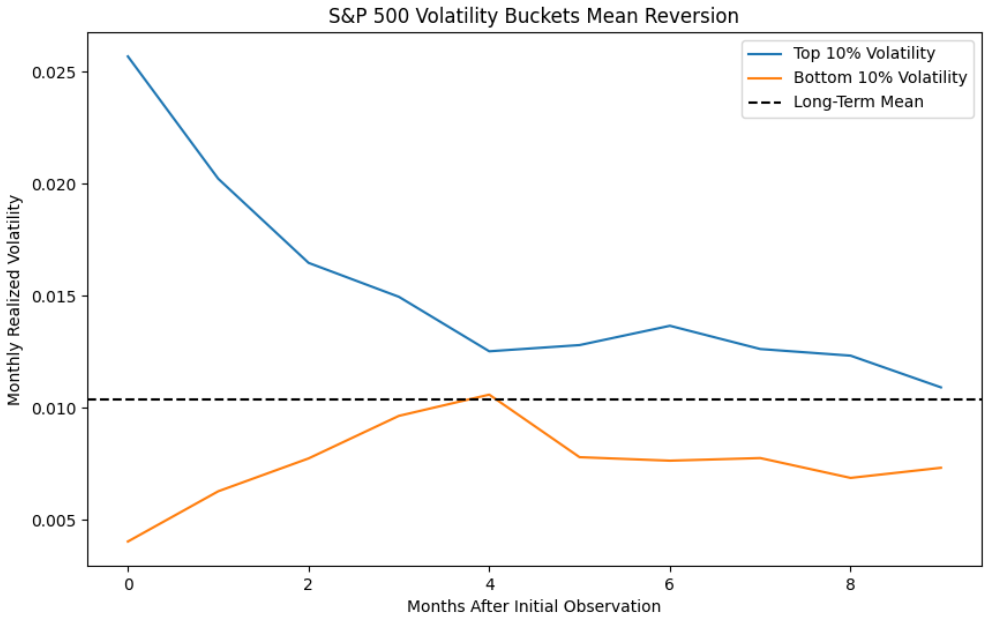


Figure 1.4: Convergence of the two baskets toward the long term mean.

Figure 1.4 is much more useful to illustrate the mean-reverting character of volatility. We clearly see that when R.V. is at its highest, it tends to go back fairly quickly towards its long term mean and to stick around it afterwards.

### 1.2.3 Negative relationship between equity returns and volatility changes

The last well-known stylized fact about realized volatility links its changes and the changes of daily returns. It has been observed, first by Black (1976), that the greater the daily change in the S&P then the smaller the change in realized volatility. Stated another way, realized volatility exhibits a negative relationship with daily returns.

This relationship is often referred to as the volatility-risk premium, as investors demand a higher expected return for taking on the additional risk associated with higher volatility.This means that when volatility is high, investors may require a higher return to compensate for the added risk, which can lead to lower actual returns.

To visualize this, we decide to look at the evolution of the daily returns between the VIX (INDEXCBOE: VIX) and the S&P500. Because of the obvious relationship between returns and volatility, the Volatility Index is a better proxy to estimate the relationship between these two components. Indeed, the VIX represents an index of the implied volatility of 30-day options on the S&P 500 Index and consequently derives its value directly from the market, not from past observations.

Even though knowing the exact pricing formula of the VIX is not primordial in our case, it does matter to have some concrete idea of how the pricing is done. To begin with, the CBOE uses two sets of options: the first is a set of near-term options with a remaining time to expiration of 30 days or less, and the second is a set of options with a remaining time to

expiration of 60 days or less. Then the implied volatility for each of these options is extracted from the market, and a weighted average is taken using the option's bid price (the weight given to each option is proportional to the option's inverse squared bid price). Finally, the weighted average is then interpolated and extrapolated to obtain the implied volatility for a hypothetical option with a 30-day expiration.

We now obtain the following graph between the $1^{st}$ of January 2000 and the $10^{th}$ of March 2023:

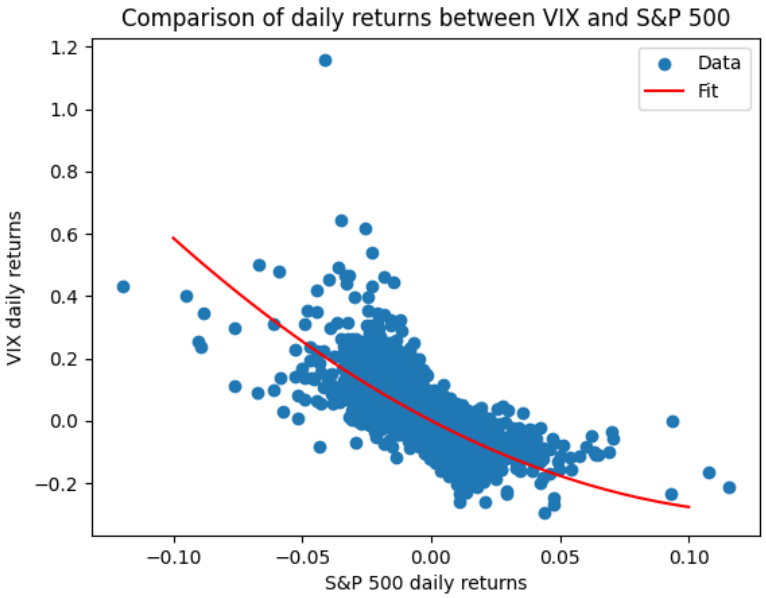


Figure 1.5: Convergence of the two baskets toward the long term mean.

The red line has been obtained by fitting a quadratic function to the data, using the non-linear least squares method. It clearly exhibits a negative relationship between the daily returns of the VIX and the daily returns of the S&P500, which would need to be taken into account when we build our model.

This empirical fact has been widely studied in the financial literature, and some of the reasons advanced suggest that higher volatility may lead to greater hedging costs and thus a lower expected return on the stock market (Campbell and Shiller, 1998), or that investors demand a higher risk premium to hold stocks when the expected variance of returns is high (Bollerslev, 2015).

# Chapter 2

# Modeling realized volatility

## 2.1 The importance of choosing the right volatility

For several volatility trading desks, shorting volatility while being delta-hedged is a ludicrous but risky strategy, aiming at making a profit from option judged as mispriced. Let us consider the classic Black Scholes framework, where hedging is supposed to be continuous and costless, among other things. In practice, this is not the case and most traders know that there is often a trade-off between being perfectly delta-hedged and the price to pay to the market to do so. Thus, it is extremely important to correctly assess the amount of shares $\Delta$ to buy in order to hedge the position. In Black-Scholes framework, delta can be computed as :

$$\Delta = N(d_1) \tag{2.1}$$

with

$$d_1 = \frac{ln\left(\frac{S_0}{K}\right) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \quad \text{and} \quad d_2 = d_1 - \sigma\sqrt{\tau}$$

as we've seen previously.

While some of these parameters are fixed by the market (strike, asset price, ...), the volatility $\sigma$ to hedge with remains the choice of the trader. Ahmad (2005) built on the work of Henrard (2003) and Carr (2005) to illustrate the impacts of this choice, knowing that it is possible to hedge with implied volatility, realized volatility, or whatever measure in between.
To realize what are the potential effects of hedging with one volatility instead of another, let us consider the previous model:

$$dS = \mu S dt + \sigma_r S dB \tag{2.2}$$

The option is priced at the implied volatility $\sigma_i$, we will therefore denote $V_i$ its price. In the same way, we note $\sigma_r$ our anticipated realized volatility and the price $V_r$ the option should be worth under this measure. Henrard (2003) and Carr (2005) elegantly demonstrated the present value of the total profit using a delta-hedged strategy. The result is as follows and gives ways to immediate interpretations:

$$PV_{profit} = V(S, t, \sigma_r) - V(S, t, \sigma_i) + \frac{1}{2}(\sigma^2 - \sigma_r^2) \int_{t_0}^{T} e^{-r(t-t_0)} S^2 \Gamma_r^2 dt \tag{2.3}$$

where $\sigma$ = actual realized volatility over the life of the trade.
In a first time, we can notice that if we $\Delta$ hedge using the implied volatility from the market, then the total profit is reduced to the integral on the right side of equation (3.3) which signs depends solely on the difference between the two considered volatilities.
On the other hand, if our forecast was correct then the total profit consists on the difference of prices between the "correct" price of the option, i.e. what it should have been, and what it was priced at the time we entered the trade.

When hedging with a different volatility, several scenarios can arise depending on how the actual volatility locates itself relatively to the implied one. The key takeaway from Ahmad's work (2005) is that the expected profit is much higher when the A.V. is different than the I.V. Even though in all cases the expected profit is insensitive to the chosen hedging volatility, the standard deviation of the profits as well as the min/max profits do depend strongly on it, which makes it crucial to assess and evaluate.

## 2.2 Modeling realized volatility

### 2.2.1 First intuition with the GARCH model

Introduced by Bollerslev (1986) as an extension of the ARCH model, the Generalized AutoRegressive Conditional Heteroscedasticity (GARCH) model has been widely used in the field of financial econometrics for forecasting the realized volatility of asset returns. This model relies on several strong assumptions, such that the returns of the asset are normally distributed, the returns are stationary, the conditional variance of the returns is time-varying and depends on the past values of the returns. Its general formulation denoted with parameters $p$ and $q$ is as follows:

$$\sigma_t^2 = \omega + \sum_i^q \alpha_i r_{t-i}^2 + \sum_j^p \beta_j \sigma_{t-j}^2 \tag{2.4}$$

The most popular GARCH model is the GARCH(1,1) which is equivalent to an ARCH($\infty$) and has the advantage of being simple to use: once $\omega$, $\alpha$ and $\beta$ have been estimated (through the O.L.S. method for example), then the returns & volatility of the past period are the only necessary elements to make a forecast.
Even though the GARCH models takes into account the mean reverting and the clustering aspects of volatility, it is not well suited for high frequency data given the assumptions it relies on. For example, high frequency data may exhibit intraday patterns such as seasonality, periodicity, and day-of-the-week effects, which are not well captured since one of the main assumptions of the model is the stationarity of the variance.

As a result, other models such as HAR (Heterogeneous Autoregressive) or HEAVY (Harmonically Weighted GARCH) models have been proposed to address the limitations of the GARCH model for high frequency data. These models incorporate additional terms to account for the complex patterns of volatility in high frequency data and have been shown to provide more accurate forecasts of the realized volatility for high frequency data.

### 2.2.2 State of the art: HAR models

Initally proposed by Corsi(2009), the HAR model owes its popularity to its simplicity and its good overall performances. It specifies R.V. as a linear function of daily, weekly and monthly R.V. components and can therefore be expressed mathematically as:

$$RV_t = \beta_0 + \beta_1^{daily} RV_{t-1}^{daily} + \beta_2 RV_{t-1}^{weekly} \beta_3 RV_{t-1}^{monthly} + \epsilon_t \tag{2.5}$$

where $\beta_1$, $\beta_2$, $\beta_3$ are unknown parameters usually estimated via the O.L.S. (Ordinary Least Squares Method). The fundamental concept of the model posits that not all agents in a market are uniform, and that they operate on varying time horizons. This heterogeneity results in varying reaction times to news, and consequently leads to different volatility components.

This model has been widely studied during the past years and several studies have attempted to add variables in order to catch and better capture the empirical facts we know

about volatility. We can think of Kambouroudis et al., (2020) who incorporated the leverage effect, overnight returns as well as the volatility of realized volatility in their set of exogenous variables, or to Bollerslev et al.(2016) who extended the classic H.A.R. model by taking into account the integrated quarticity derived from realized volatility. A broader formula for this model would therefore be written as:

$$RV_t = \beta_0 + \beta_1^{daily} RV_{t-1}^{daily} + \beta_2 RV_{t-1}^{weekly} \beta_3 RV_{t-1}^{monthly} + \alpha' X' + \epsilon_t \tag{2.6}$$

where $X$ is a $k$-dimensional vector of exogenous variables (i.e., $X \in \mathbb{M}_{k,1}$)

The same way there is no consensus on the most accurate estimator of volatility, there is not consensus either on the best performing HAR models according to Audrinos and Knaus (2016), since the relevance of lagged realized variances and added variables is strongly dependent of the market conditions.
However across all the major studies that have been done on the subject, some variables emerge due to their strong predictive power, such as implied volatility of the market (e.g., the VIX) and overnight returns (Kambouroudis et al., 2020), as well as more recently the volatility near the close (Zhang et al., 2022).

The method used to determine these coefficients is subject to a great deal of discussions in the scientific community given the numerous flaws of the O.L.S. and that is why we will study different ways to evaluate these coefficients.

## 2.3 Estimating the coefficients

It has been commonly done in the literature to estimate the coefficients of the H.A.R. model with the Ordinary Least Squares method (O.L.S.). However given the strong assumptions on which this method relies (homoscedasticity, no auto-correlation, ...) and the now well-known facts about R.V. (strong outliers, non normality, ...), other methods have been developed to face these issues. That being said, it is still worth examining the ground on which these estimators have been built and to remind ourselves of a widely used estimation method.

### 2.3.1 Ordinary Least Squares Method

As previously stated, this method requires strong assumptions which are hard to satisfy for time series. Among those, homoscedasticity is particularly hard to verify since the distribution of the error terms often exhibits skewness and heavy tails, and the phenomenon of volatility clustering goes against the assumption of i.i.d. error terms.
Considering the model from equation 3.6, we are aiming at minimizing the sum of the squared residuals, i.e., solving the following optimization problem:

$$min \sum_{23}^{n} \left( RV_t - (\beta_0 + \beta_1^{daily} RV_{t-1}^{daily} + \beta_2 RV_{t-1}^{weekly} \beta_3 RV_{t-1}^{monthly} + \alpha' X + \epsilon_t) \right)^2 \tag{2.7}$$

This problem can easily be solved using a more compact matrix writing where, if we aggregate our explanatory variables into one matrix $X$, equation 3.6 becomes:

$$Y = X\beta + \epsilon \tag{2.8}$$

and the solution to the O.L.S problem can conveniently be written as:

$$\hat{\beta} = (X'X^{-1})X'Y \tag{2.9}$$

The loss function used here results in an arithmetic mean-unbiased estimator but it is worth to note that similar calculations with different loss functions have been studied over the years.

To quote a few, an absolute-value loss function is associated with the famous Least Absolute Deviations (L.A.D.) method which is less sensitive to outliers whereas a mixed of quadratic and linear functions is associated to the Huber loss function, mathematically defined as:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2, \text{for } |a| \leq \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), \text{otherwise.} \end{cases} \tag{2.10}$$

The difference in attitude regarding outliers can be illustrated by plotting the values taken by the different loss functions:

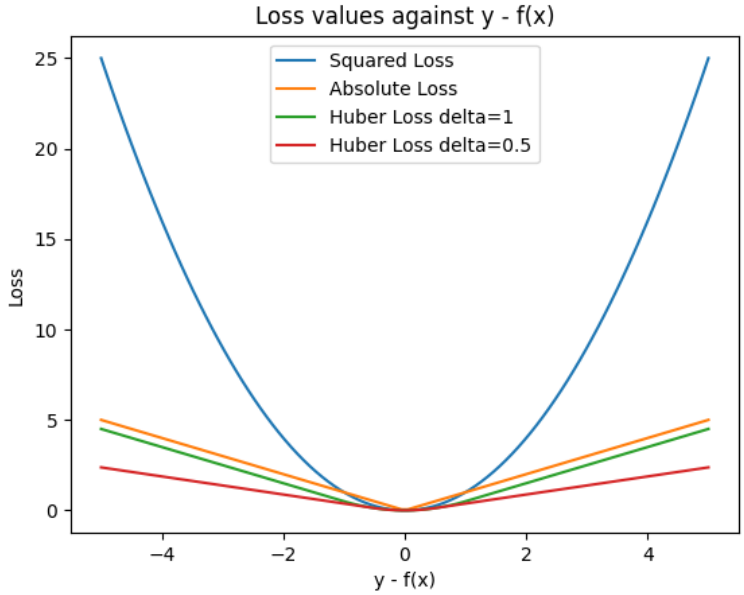


Figure 2.1: Illustration of the different loss functions.

Even though elegant and convenient to use in practice to estimate coefficients in a regression, this method has progressively been augmented to overcome the issues previously described.

### 2.3.2 Non Negative Least Squares Method

Estimating the coefficients of the H.A.R. model can be challenging due to the presence of non-negative constraints on the parameters, given that volatility can not be negative. The Non Negative Least Squares (N.N.L.S.) method is a powerful technique that can overcome this issue by finding the non-negative coefficients that minimize the sum of squared errors between the actual and predicted values of realized volatility. Using N.N.L.S. in the context of forecasting realized volatility can help to ensure that the estimated coefficients of the H.A.R. model are non-negative and thus, meaningful and interpretable in the financial domain.
The optimization problem to solve is the same as the one defined for the O.L.S. method, except that a constraint is imposed on the resulting set of coefficients.

### 2.3.3 Weighted Least Squares Method

To manage the shortcomings associated with the O.L.S., the Weighted Least Squares (W.L.S.) method has often been used to produce volatility models and is an estimation technique which weights the observations proportional to the reciprocal of the error variance for that observation and so overcomes the issue of non-constant variance. The main idea is to give small weights to

observations associated with higher variances to shrink their squared residuals.
Using the more general equation 3.8, the optimization problem we now have to solve becomes:

$$min\left((Y - X\hat{\beta})'W(Y - X\hat{\beta})\right) \tag{2.11}$$

where $W$ is a diagonal matrix with elements equal to $w_1, ..., w_n$.
The estimator can then be found by deriving this equation and applying basic algebra rules to find:

$$\hat{\beta} = (X'WX)^{-1}X'WY \tag{2.12}$$

## 2.4 Extending the model to non-linear effects

With the improvement of processing power over the last decades, more resource-consuming techniques have been developed and used in the statistical field. Some of the main breakthroughs appeared in the field of Machine Learning (M.L.), which has been more and more intensively used in Quantitative Finance to solve complex problems. Several models are now capable of inferring by themselves relationships between variables, detecting patterns in complex data and have strong predictive power with out-of-sample data.
In this master thesis, we decided to focus on two techniques: the Gaussian Process Regression which belongs to the supervised M.L. branch and XGBoost models, due to their strong forecasting performances and their abilities to capture non-linearities.

### 2.4.1 Gaussian Process Regression (G.P.R.)

Gaussian Processes belong to stochastic processes and have been widely used to solve regression and classification problems. Contrarily to many supervised M.L. models that learn exact values for every input parameter, the G.P. approach infers a probability distribution across all possible values. Let us first begin with the basic concepts to understand the G.P.R. method.
First and foremost, a Gaussian Process is defined as a collection of random variables $X_i$ such that any subset of these variables is jointly Gaussian:

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} \sim \mathcal{N}(\mu, \Sigma) \tag{2.13}$$

where $n$ is the number of random variables in the subset, $\mu = E(X) \in \mathbb{R}^n$ and $\Sigma = Cov(X_i, X_j) = E\left((X_i - \mu_i)(X_j - \mu_j)'\right) \in \mathbb{M}_{n,n}$
The density function of such a random variable, which is called a Multivariate Normal Distribution (M.V.N.) is expressed as:

$$\mathcal{N}(x|\mu, \Sigma) = \frac{exp(-\frac{1}{2}(x - \mu)^T\Sigma^{-1}(x - \mu))}{\sqrt{(2\pi)^D det(\Sigma)}} \tag{2.14}$$

where $\mu = E(x) \in \mathbb{R}^D$ and $\Sigma = Cov(x) \in \mathbb{M}_{D,D}$.
To get a better understanding of the evolution of such a function, we can observe it for two variables, i.e., for a Bivariate Normal Distribution (B.V.N.):

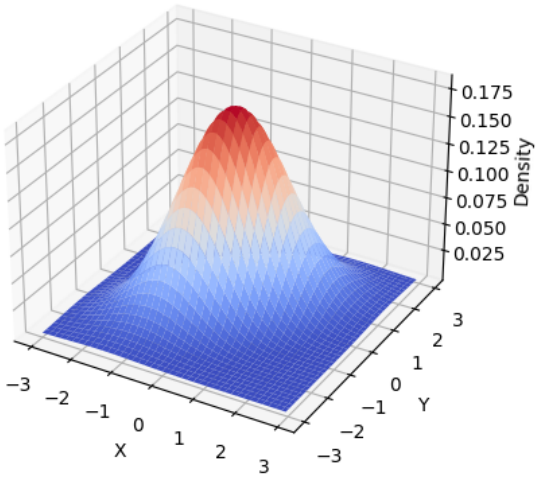


Figure 2.2: Probability density function for a B.V.N.

Essentially, Gaussian Processes can be thought of as a generalization of the Gaussian Probability distribution to infinitely many variables. A Gaussian Process can therefore be seen as a Gaussian random function specified according to:

$$f(x) = \mathcal{GP}(m(x), k(x, x')) \tag{2.15}$$

where $x \mapsto m(x)$ is the mean function and $(x, x') \mapsto k(x, x')$ is the covariance function, also known as kernel.
Most of the times, the mean function is assumed to be 0 for the sake of simplicity given the linear nature of Gaussian Processes. If this assumption were not to hold, then correcting it by adding the mean would be fairly easy to do. Picking an appropriate form for the kernel function reveals itself to be a more challenging matter as these functions can take every possible form as long as they are semi-positive, definite and symmetric. More importantly, the choice of the kernel function reflects our prior knowledge of the form of the function we are modeling since the kernel depicts how much influence the $i$-th and $j$-th point have on each other. Some of the most commonly used kernels are known as the Radial Basis Function (R.B.F.) and the periodic kernel, which corresponding equations are as follows:

$$K(x_i, x_j) = \begin{cases} \sigma^2 \exp\left(-\frac{\|x_i - x_j\|^2}{2l^2}\right), & \text{R.B.F. kernel} \\ \sigma^2 \exp\left(-\frac{2\sin^2\left(\frac{\pi\|x_i - x_j\|}{p}\right)}{\ell^2}\right), & \text{Periodic kernel} \end{cases} \tag{2.16}$$

where $\sigma$ determines the average distance away from the function's mean, $l$ impacts the reach of influence on neighbors and $p$ is the periodicity parameter. The following figure shows an example of of a covariance matrix for both kernels, where $\sigma = 1$, $p = 1$ and $l = 1$.

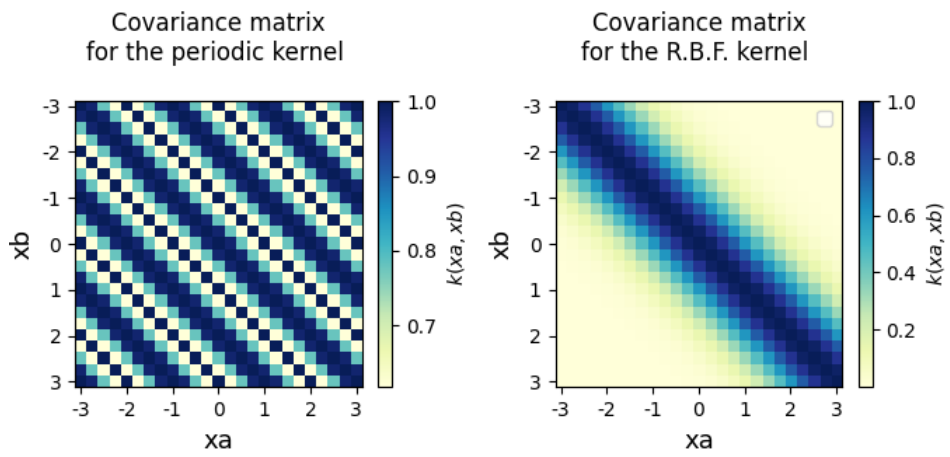


Figure 2.3: Example of covariance matrixes for the R.B.F. & Periodic kernels.

The theoretical layout having been set, let us on now focus on how to train and make predictions with G.P.R. Assuming a set of data-points $X_1$ for which we have observed the values $Y_1$, we want to make predictions of $Y_2$ for a set $X_2$, which is equivalent to computing $p(Y_2|Y_1, X_1, X_2)$. We call $Y_1$ the testing dataset and $Y_2$ the training dataset. It is hardly possible to consider the training set $Y_2$ to be perfect given error measurements for example and it is therefore a common practice to model the $Y_2$ as noisy realizations of the G.P. $f$. Fortunately, we can model this uncertainty by adding an error term $\epsilon$ to each of our training points. This way, we have:

$$Y_2 = f(Y_1) + \epsilon \tag{2.17}$$

where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$
By definition of a Gaussian Process, we know that $Y_1$ and $Y_2$ are jointly Gaussian and can therefore write:

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k(X_1, X_1) + \sigma_n^2 I & k(X_1, X_2) \\ k(X_2, X_1) & k(X_2, X_2) I \end{bmatrix} \right) \tag{2.18}$$

The prediction step consists in estimating the mean and the variance for $Y_2$, which is doable using the characteristics of joint Gaussian vectors. We can express these values as:

$$Y_2|Y_1 \sim \mathcal{N}(\bar{f}, Cov(f))$$
$$\text{with} \begin{cases} \bar{f} = k(X_1 X_2)' \left(k(X_1, X_1) + \sigma_n^2 I)\right)^{-1} Y_1 \\ Cov(f) = k(X_2, X_2) - K(X_1, X_2)' \left(K(X_1, X_2 + \sigma_n^2 I)\right)^{-1} K(X_1, X_2) \end{cases} \tag{2.19}$$

The mean value $\bar{f}$ represents the best estimate of $Y_2$ while $Cov(f)$ measures the uncertainty associated with this estimate.
As seen previously, the kernel can be a function of various parameters ($\sigma$, $p$, $l$, ...) which are called hyperparameters that we will denote $\theta$. Part of the learning process relies on tuning these hyperparameters based on the data since it is highly unlikely that we have a concrete idea of the values these should take. To do so, we are aiming at maximizing the marginal likelihood $p(Y|X, \theta)$ of the G.P. distribution based on the observed data $X, Y$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left(p(\mathbf{y} \mid X, \theta)\right) \tag{2.20}$$

where $p(Y|X, \theta)$ was given in equation 3.14.
Given the strict monotonic nature of the log function, we can get rid of the exponent and maximize the log marginal likelihood:

$$\log p(Y \mid X, \theta) = -\frac{1}{2}(Y - \mu_\theta)' \Sigma_\theta^{-1} (Y - \mu_\theta) - \frac{1}{2} \log |\Sigma_\theta| - \frac{d}{2} \log 2\pi \tag{2.21}$$

where the first term represents the data-fit and the other one is a complexity penalty. We can afterward derive this equation to obtain:

$$\begin{aligned}\frac{\partial}{\partial\theta}\log p(Y|X;\theta) &= \frac{1}{2}Y'k(X,X)^{-1}\frac{\partial k(X,X)}{\partial\theta}k(X,X)^{-1}Y - \frac{1}{2}\mathrm{Tr}\left(k(X,X)^{-1}\frac{\partial k(X,X)}{\partial\theta}\right)\\ &= \frac{1}{2}\mathrm{Tr}\left(\alpha\alpha' - k(X,X)^{-1}\right)\frac{\partial k(X,X)}{\partial\theta}\end{aligned} \tag{2.22}$$

where $k(X,X)$ is a kernel matrix, $\alpha = k(X,X)^{-1}y$, and Tr denotes the trace of a matrix. These derivatives are then used in methods such as the gradient ascent (e.g., in Newton method) to obtain the best set of hyperparameters.

### 2.4.2 XGBoost

Standing for Extreme Gradient Boosting and developed by Tianqi Chen from the University of Washington in 2016, this machine learning tool has been one of the best performers during Kaggle competitions over the past several years. It is well known for its superiority compared to more traditional techniques as well as the fastness at which the model is trained (Chen and Geuestrin, 2016) More specifically, XGBoost is a supervised machine learning technique which aims at predicting a target variable $y_i$ given a set of training data with multiple features $x_i$ just as linear regression for example. It relies on the gradient boosting method which uses additive modeling to gradually nudge a first draft of a models toward a good model, by gradually adding submodels.
For a given set of data with $n$ observations and $m$ features $\mathcal{D} = (x_i, y_i)$, we can therefore write the prediction made $\hat{y}$ as the addition of $K$ weak models $f_k(x_i)$, such that:

$$\hat{y_i} = \sum_{k=1}^{K} f_k(x), f_k \in \mathbf{F} \tag{2.23}$$

where $\mathcal{F} = \left\{f(\mathbf{x}) = w_{q(\mathbf{x})}\right\}\left(q : \mathbb{R}^D \to K, w \in \mathbb{R}^K\right)$ represents the space of Classification and Regression Trees (C.A.R.T.). More specifically, $T$ is the number of leaves in the tree, $q$ represents the structure of each tree and each $f_k$ is the prediction from each tree. XGBoost can then be seen as a collection of decision trees, where each tree contains a continuous score on the $i$-th leaf represented by $w_i$.
As in most M.L. algorithms, we need to optimize a loss function $L$ between observed data and predicted data in order to train the model. We presented several of these functions previously, but is important to recall that there exists an infinite number of them. In addition to the loss function, we also need to introduce a regularization term in order to prevent overfitting in our model. There exists many additional ways to prevent this to happen that we will present later on. In XGBoost algorithm, the regularization penalty is defined as:

$$\Omega = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \tag{2.24}$$

where $\gamma$ is a user definable penalty meant to encourage pruning, a machine learning technique that compresses decision trees by removing non-critical and redundant sections by comparison of the gain from an added branch with the constant $\gamma$, while $\lambda$ defines the emphasis we put on the regularization penalty. We can finally write the objective of the model, i.e., the function to minimize as:

$$Obj^{(t)} = \sum_{i=1}^{N} L\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^{t}\Omega\left(f_i\right) = \sum_{i=1}^{N} L\left(y_i, \hat{y}_i^{(t-1)} + f_t\left(x_i\right)\right) + \sum_{i=1}^{t}\Omega\left(f_i\right) \tag{2.25}$$

Since it is intractable to learn all the trees at once and an additive strategy presented earlier is used, the gradient descent technique is employed here to optimize the objective, which consists into computing: $\partial_{\hat{y}} \operatorname{Obj}(y, \hat{y})$ at each iteration, then improving the forecast $\hat{y}$ along the direction of the gradient to minimize the objective.
The objective function in 3.25 can be simplified using Taylor's development at the second order, which yields:

$$Obj^{(t)} \simeq \sum_{i=1}^{N} \left[ L\left(y_i, \hat{y}^{(t-1)}\right) + g_i, f_t\left(x_i\right) + \frac{1}{2} h_i f_t^2\left(x_i\right) \right] + \sum_{i=1}^{t} \Omega\left(f_i\right) \tag{2.26}$$

where: $g_i = \partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$ We can remove the constant terms since this equation is due to be differentiated and to make it more readable, we finally obtain:

$$\widetilde{Obj^{(t)}} = \sum_{i=1}^{n} \left[ g_i f_t\left(x_i\right) + \frac{1}{2} h_i f_t^2\left(x_i\right) \right] + \Omega\left(f_t\right) \tag{2.27}$$

Knowing a tractable form of our objective function at the $t$-th step, we are looking to find $f_t$ in order to optimize it. To do so, XGBoost relies on decision trees, with one of the main challenges being to find a tree which improves the prediction along the gradient. Using the same notations as previously employed, a tree is defined as:

$$f_t(x) = w_{q(x)} \tag{2.28}$$

with $q(x)$ a function assigning every data-point to the $q$-th leaf and $w_{q(x)}$ being the score assigned on the $q$-th leaf. Knowing this definition, the objective function can be even further simplified as:

$$\widetilde{Obj^{(t)}} = \sum_{j=1}^{T} \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \tag{2.29}$$

where $I_j = \{i \mid q\left(x_i\right) = j\}$ is the index set containing the indices of data points assigned to the $j$-th leaf.
We can easily compute the optimal weight $w_j^*$ from this equation by differentiating it, which leads to:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{2.30}$$

This weight can then be replaced in our objective function to obtain:

$$\widetilde{Obj^{(t)}} = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{2.31}$$

This expression is crucial since it is used in XGBoost algorithm do determine the relevance of a tree, and is equivalent to what is known as the impurity score in more classic trees.

Now that we have a more precise idea of two of the best performing M.L. tools used in this branch of time-series forecasting, let us explore the details behind hyperparameter-tuning since they influence a lot the performance of a model and have been the center of optimization in this field.

### 2.4.3 Hyperparameters tuning

Hyperparameter tuning is a critical aspect of machine learning (ML) that can have a significant impact on the performance of a model. Machine learning algorithms rely on a set of parameters that are optimized during training to minimize the error between the predicted and actual values. For example, Gaussian Process Regression relies on the kernel function, the length-scale parameters $l$, the noise variance $\sigma_n^2$ while XGBoost uses the learning rate, the number of trees $M$, the regularization parameters $\gamma$ and so on.

However, hyperparameters, which control the behavior of the model during training, cannot be optimized in the same way. Instead, they must be carefully selected and fine-tuned to achieve the best possible performance. The proper selection and tuning of hyperparameters can significantly improve the accuracy of a model and ensure its ability to generalize to new data. This topic has been extensively studied in the field of ML, and several techniques have been proposed to automate the process of hyperparameter tuning. It is extremely important to make a clear distinction between hyperparameters and model parameters. On the one hand model parameters are learned from the training data when the objective function is optimized while hyperparameters define the actual structure of the model. The following schema illustrates the difference between the two:

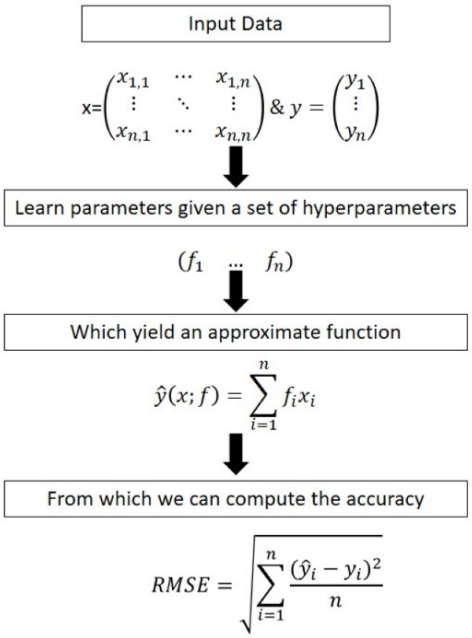


Figure 2.4: From hyperparameters to parameters.

Parameters are learned from a given set of hyperparameters, and there is no literal formula to compute the most optimal set of hyperparameters. In this section we will focus on the three most commons techniques, known as Grid Search, Random Search and Bayesian Optimization.

#### 2.4.3.1 Grid Search

Grid search is perhaps the first and more intuitive idea when it comes to hyperparameter tuning. It is an exhaustive algorithm that spans all the combinations for a given set of hyperparameters, where each parameter belongs to a given range. If we have $n$ hyperparameters, and each $i$-th hyperparameter can take $m_i$ values, then the number of models that would be built in this method is $\prod_{i=1}^{n}(m_i)$. It is an effective method when the search space is limited but it can

rapidly become time and resource-consuming and the scalability for larger models is a real issue.

#### 2.4.3.2 Random Search

This method is close to Grid search, except that the range of values for hyperparameters is not set in stone, but we rather define probability distributions on ranges of values for each of them. This approach is based on the assumption that in most cases, hyperparameters are not uniformly important. The log-uniform distribution is commonly used in this method when the range of interest for the hyperparameter spans multiple orders of magnitude. This is still an expansive method to perform but it presents the advantage to detect more easily the import values for hyperparameters compared to Grid search, as illustrated below:

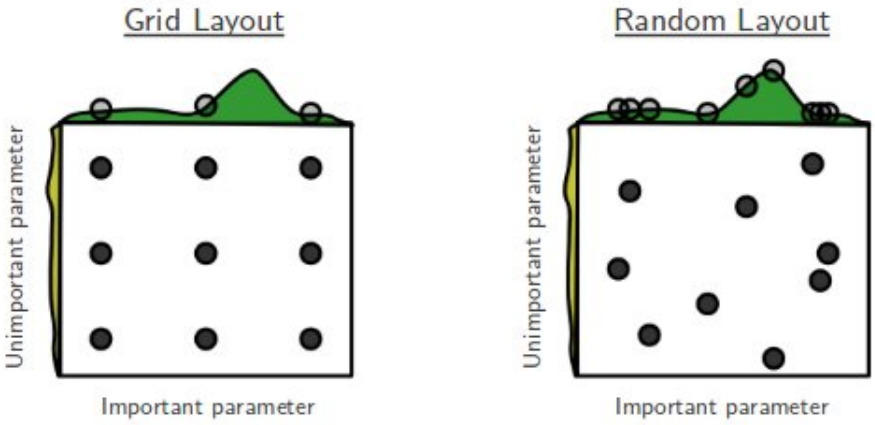


Figure 2.5: Illustration of Grid search and Random search sampling. Source: Big Data analytics in medical imaging using M.L.

As we can see, by sampling over more unique couples for two hyperparameters, this technique is able to better catch the important values.

#### 2.4.3.3 Bayesian Optimization

The two methods presented previously work well and are often the go-to choice of many M.L. optimization problems. The main issue with them is their time complexity given that they require the training of a model at every iteration, to which we often need to add cross-validation. In this sense, they do not learn from the previous trained models and do not make educated guesses related to what the best set of hyperparameters might be, and this is why the Bayesian Optimization technique arised.
When it comes to hyperparameter tuning, the functions to evaluate are whole models which might take a lot of time to train (training a small neural network with 10 hidden units will take less time than a bigger network with 1000 hidden units, but it remains a sizeable amount). At the core of Bayesian Optimization is the creation of a surrogate function that approximates the objective function being optimized, and an acquisition function that directs the search towards promising regions of the hyperparameter space. The surrogate function is typically modeled as a Gaussian Process (Adams et al. 2012), which allows for uncertainty quantification and the incorporation of prior knowledge. GPs are a flexible and interpretable way of modeling complex functions, and provide a posterior distribution over the objective function at any point in the hyperparameter space. The posterior distribution can be used to compute the acquisition function and to guide the search towards promising regions of the hyperparameter space. The

surrogate function can be updated after each evaluation of the objective function, incorporating the new information and refining the posterior distribution.

The acquisition function balances exploration and exploitation by trading off between regions of high uncertainty and regions of high expected improvement. The most commonly used acquisition functions are Expected Improvement (E.I.), Probability of Improvement (P.I.) or GP Upper Confidence Bounds (G.P.U.C.B.). Following the work of Klein and Falkner (2017), we decide to use E.I. for its robustness and its use of global information in contrast with P.I. and G.P.U.C.B. If we assume that our optimization problem is optimizing $\arg\max_x f(x)$ and that the current best is at $x^+ = \arg\max_{x_i \in (x_j)_{j\in[1:n]}} f(x_i)$., then the improvement function is defined as:

$$I(x) = \max\{0, f(x) - f(x^+)\} \tag{2.32}$$

and therefore leads to the Expected Improvement function which is our acquisition function:

$$EI(x) = arg\max_x E(I(x)|\mathcal{D}) \tag{2.33}$$

where $\mathcal{D} = (x_i, y_i)_{i\in[1:n]}$.
A closed form for this function can be obtained as:

$$\mathbb{E}(I(\mathbf{x})) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+))\,\Phi(z) + \sigma(\mathbf{x})\phi(z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \tag{2.34}$$

where $z = \frac{\mu(x) - f(x^+)}{\sigma(x)}$, $x \mapsto \Phi(x)$ is the C.D.F. of a standard normal distribution and $x \mapsto \phi(x)$ is its P.D.F.
Having set the ground on the maths behind Bayesian Optimization, we can now study how they are implemented in practice through the algorithm:

**Algorithm 1** Bayesian Optimization Algorithm

1: **Input:** Set of data $\mathcal{D}_0$.
2: **Output:** Sequence $\{(x_n^*, y_n^*)\}_{n=1}^T$.
3: **for** $n = 1, 2, ..., T$ **do**
4: Fit probabilistic model on data $\mathcal{D}_{n-1}$
5: Find $x_n^* = \arg\max_x E(I(x)|GP(D))$ of (11).
6: Evaluate the objective function: $y_n^* = f(x_n^*)$.
7: Augment the observation set $\mathcal{D}_{n+1} = \mathcal{D}_n \cup \{(x_n^*, y_n^*)\}$.
8: **end for**

In practice, the algorithm is typically run until a termination criterion is met, which can be either a fixed number of iterations or a convergence threshold on the expected improvement of the acquisition function. The termination criterion should be chosen based on the application requirements and the available resources.

It is worth noting that the convergence rate of Bayesian optimization is usually slower than that of other optimization methods such as gradient-based methods, but it is well-suited for optimizing expensive, black-box functions where direct evaluation is difficult or time-consuming.

# Chapter 3

# Implementation and results

## 3.1 Data Description

Given the non-negligible costs associated to high-frequency data, this master's thesis will be focusing on daily data to predict the next day daily realized volatility. If more time was allocated as well as more resources, it would be extremely interesting to purchase a dataset from Lobster or FirstRateData for example in order to exploit the power of intraday data, as it has been done in many previous studies (see Zhang et al. 2023). Given that the intraday volatility can be seen as an aggregation of the high-frequency data, we define our own measure of volatility for this study, which serves as a proxy for the commonly used estimator defined in Chapter 2. We therefore write:

$$RV_t = \left| \log \left( \frac{\text{close}_t}{\text{open}_t} \right) \right| \tag{3.1}$$

We thought it would be more relevant to focus on the features selection as well as model improvements rather than the quality of data, given that it is easily accessible in the professional world but much less so as a student. Ours comes from the Yahoo Finance API, an easily accessible source of data using the Python package `yfinance`. Note that given material constraints the code used is written with Pycharm and available as .py files on Github but has been ran on Google Collab as .ipynb ones.
Following the same principles as Christensen et al. (2021) regarding the universe of stocks we study at, we pick the fifteen stocks with the highest market capitalization which have been in the S&P500 since January 1, 2001 through March 3, 2023. The consequent numbers of crisis which happened during this time period will enable them to train them on this data (internet bubble of early 2000's, global crisis of 2008, European debt crisis of 2010) as well as to see how they perform during them (Covid crisis of 2020-2021).

Overall our data is comprised of 5 828 dates for each of the fifteen stocks, that we splitted according to the classic 80-20 rule to train and test our models. Below is a short summary of the universe of stocks we are considering , as well as some statistics computed about them:

| Symbol | Min log close price | Max log close price | Mean log close price | Min daily volatility | Max daily volatility | Mean daily volatility |
|---|---|---|---|---|---|---|
| AAPL | -1.451213 | 5.204062 | 2.124468 | 0.0 | 0.127887 | 0.014353 |
| ADBE | 2.122262 | 6.534326 | 4.107826 | 0.0 | 0.222408 | 0.015183 |
| AMD | 0.482426 | 5.087041 | 2.514686 | 0.0 | 0.223770 | 0.023741 |
| AMZN | -1.208985 | 5.228809 | 2.314915 | 0.0 | 0.253860 | 0.017113 |
| BA | 3.221273 | 6.088183 | 4.558186 | 0.0 | 0.164957 | 0.012419 |
| BAC | 1.144223 | 4.005513 | 3.191558 | 0.0 | 0.239480 | 0.013579 |
| COST | 3.304686 | 6.410257 | 4.565030 | 0.0 | 0.142715 | 0.010482 |
| HD | 2.890372 | 6.031118 | 4.259674 | 0.0 | 0.122047 | 0.011210 |
| INTC | 2.491551 | 4.315820 | 3.387273 | 0.0 | 0.127875 | 0.013149 |
| JNJ | 3.533687 | 5.225800 | 4.406436 | 0.0 | 0.084229 | 0.007138 |
| JPM | 2.737609 | 5.146215 | 4.023119 | 0.0 | 0.203488 | 0.012517 |
| MSFT | 2.718001 | 5.838051 | 3.857775 | 0.0 | 0.109290 | 0.010790 |
| NEE | 1.526396 | 4.536463 | 2.953175 | 0.0 | 0.112577 | 0.009004 |
| NVDA | -0.487488 | 5.810422 | 2.074498 | 0.0 | 0.332794 | 0.021915 |
| XOM | 3.410157 | 4.780551 | 4.185489 | 0.0 | 0.132060 | 0.009671 |

Table 3.1: Statistics regarding the universe of stocks.

As we can see, the range of values for daily realized volatility can be wide, though always lowered by 0 since it happens that all these stocks had at least one day where the open price was similar to the close price. Let us now dig into the conception of the models and the results they provided.

## 3.2 Modeling

### 3.2.1 Linear Models

We use the packages `statsmodels` and `sklearn` to build our models on each of the fifteen stocks. We fit a model for each stock and each way of determining the coefficients, using weights inversely proportional to the variance of the residuals for the Weighted Least Squared regression. All the coming plots are for the stock Apple given its predominant weight in the SP to this day, but it is easy to reproduce the results for any other ticker using the provided code. Given that the testing dataset still covers more than 3 years, we decided to plot as well the results focusing on data since 2022, given that the past year and a half markets have been fairly volatile. Only the detailed plots are represented here but the predicted values over the whole period are included in the Appendix A. We therefore obtain the following graphs:

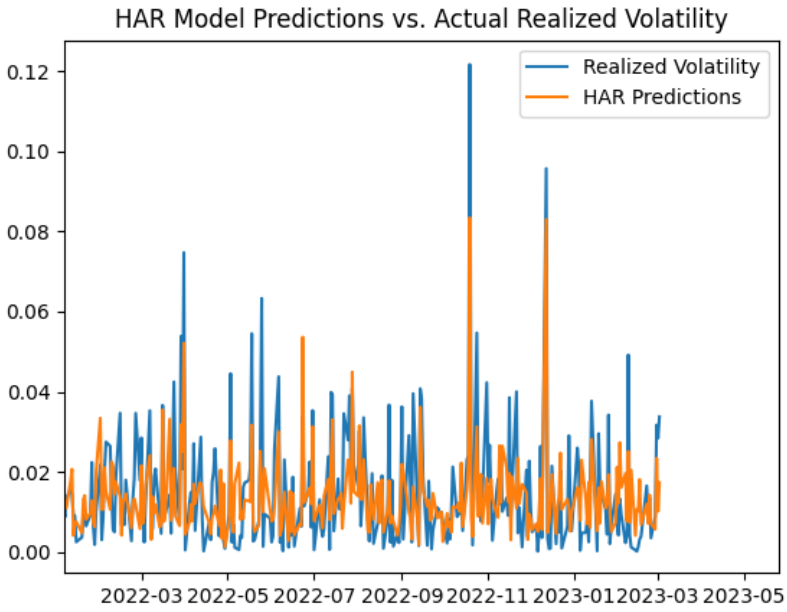


Figure 3.1: H.A.R. predictions on a subset, O.L.S.

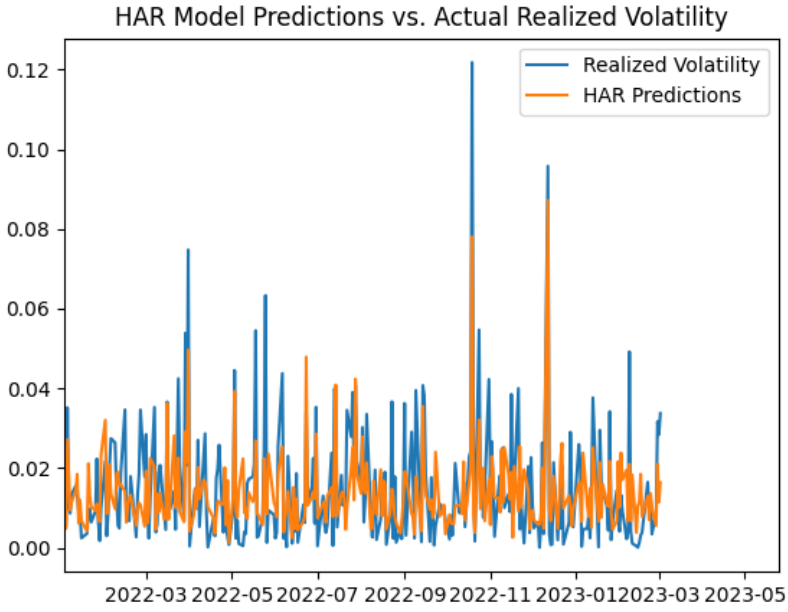


Figure 3.2: H.A.R. predictions on a subset, N.N.L.S.

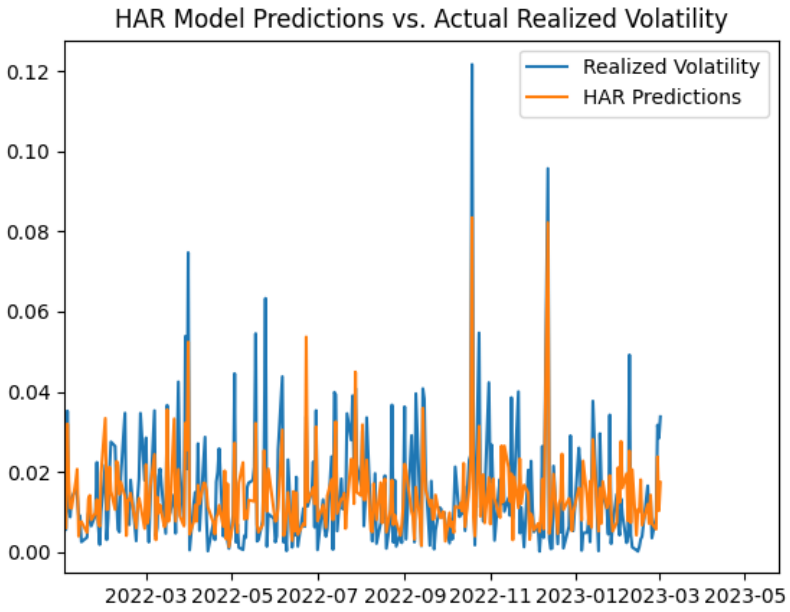


Figure 3.3: H.A.R. predictions on a subset, W.L.S.

To the naked eye, the graphs look quite similar and to a certain extent, the 3 models managed to predict the rises in volatilities occurring at the end of October 2022 and mid-December. To better assess which model performed the best, we look at the Mean Squared Errors (M.S.E.) and the less outlier-sensitive Mean Absolute Error (M.A.E.), respectively defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \text{ and } MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.2}$$

We computed these scores as the average of the scores obtained for each model, the breakdown stock by stock being available in the Appendix A as well, and obtained the following results:

| Mean MAE OLS | Mean MSE OLS | Mean MAE NNLS | Mean MSE NNLS | Mean MAE WLS | Mean MSE WLS |
|---|---|---|---|---|---|
| 0.00705154 | 0.000115467 | 0.00728083 | 0.000121066 | 0.00704021 | 0.000115187 |

Table 3.2: Averaged MAE & MSE across the universe of stocks for the linear HAR.

Despite the differences being small between each method, the W.L.S. regression seems to perform better on average compared to the O.L.S. or the N.N.L.S. By down-weighting the observations with high variance and up-weighting the ones with low variance, we have more efficient estimators which enable us to get better results. We also notice that the models estimated via the N.N.L.S. are always outperformed across all stocks by the O.L.S. and the W.L.S. The reason could be that the N.N.L.S. is less flexible in capturing the underlying volatility dynamics and therefore might not be able to capture the full complexity of the data.

We also find it interesting to observe the most significant variables in our study for the two best performing models (in the main body are only given the results for Apple's case but more detailed tables are available in Appendix A). To do so, we look at the magnitude of the coefficients as well as their associated p-values, which are given below:

| | constant | p-value | daily rv | p-value | 5 days mean rv | p-value | 22 days mean rv | p-value |
|---|---|---|---|---|---|---|---|---|
| O.L.S. | 0.0002 | 0.608 | -0.2134 | 0.000 | 1.2557 | 0.000 | -0.0593 | 0.08 |
| W.L.S. | 0.0001 | 0.635 | -0.223 | 0.000 | 1.249 | 0.000 | -0.0383 | 0.157 |

Table 3.3: Coefficients of the H.A.R. model and p-values associated for Apple.

In both cases, the most significant variables are the one day and five days lagged realized volatilities, while the 22-days R.V. is significant at 10% only for one model, which makes it hard to discard completely.

We now want to compare these results with those yielded by Machine Learning techniques and to do so we will first start with Gaussian Process Regression.

### 3.2.2 Machine Learning Models

#### 3.2.2.1 Gaussian Process Regression

As exposed in the theoretical part of the Gaussian Process Regression, we assumed that the training data had a zero mean and therefore first need to standardize the data. In addition, given that various used kernels look at the distance $\|x - x'\|$, standardizing the data is definitely an important step and not doing so could greatly compromise the results. We used the `skoptimize` package for this task and the `sklearn` one to perform the regression and tune our hyperparameters.
Following the work done by Han et al. (2016) and Liu et al. (2020), we use the following kernels:

$$K(x_i, x_j) = \begin{cases} \exp\left(-\frac{\|x_i - x_j\|^2}{2l^2}\right), & \text{R.B.F. kernel} \\ \left(1 + \frac{\sqrt{3}\|x_i - x_j\|^2}{\rho}\right) \exp\left(-\frac{\sqrt{3}\|x_i - x_j\|^2}{\rho}\right), & \text{Matern 3/2 kernel} \end{cases} \tag{3.3}$$

For both the Matern kernel and the Radial Basis Function kernel, we only tune the length-scale $\rho$ (see Rasmussen et al. 2006 page 86 for more details regarding Matern kernels), which is done during the fitting using the Maximum Likelihood method by default. It makes sense they share the same hyperparameter to tune since a Matern kernel of infinite order converges towards an R.B.F. kernel. The other parameters left to work on are $\alpha$ which corresponds to the regularization strength and $n_restarts_optimizer$ which is the maximum number of tries to optimize the set of hyperparameters. The optimization is done using a 5-fold cross validation, meaning that the original dataset is randomly partitioned into 5 equally sized subsets and one of the 5 subsets is used as the test set, while the remaining 4 subsets are used as the training set. This is done to ensure that the hyperparameters are robust and not overly specific to a particular random split of the data.
Regarding Bayesian Optimization, the acquisition function used is the Expected Improvement as specified earlier, while the objective function used in the code is the negative Mean Squared Error function (since maximizing a function is equivalent to minimizing its opposite) as is commonly used for these problems.

While the theory is elegant and sounds promising, the reality was very different. In fact, the time taken to fit a G.P.R. was too long for Google Collab and despite the author of this paper waking up every 1 hour 30 min to maintain an active session during the weekend, the code would stop arbitrarily after a few hours. We therefore decided to run it on a local machine as a last resort solution. It took between 4 hours (random search), more than 11 eleven hours (Bayesian optimization) and an undefined amount of time (grid search) to train a model and obtain semi-decent predictions with a search space narrowed iteration after iteration. Given the

technical limitations of the computer used, we only trained the model for the Apple stock and the predictions were terrible compared to classic H.A.R. models. We only had the opportunity to fit models for the Matern 3/2 kernel, for which the accuracy measures were:

| | MAE | MSE |
|---|---|---|
| Random search | 0.637 | 0.890 |
| Bayesian optim. | 0.544 | 0.566 |

Table 3.4: MAE & MSE of the GPR for the Apple stock.

These values are approximately between 100 and 5 000 times worse than the ones obtained with a classic H.A.R. model estimated with any of the linear method. This might be due to the fact that Gaussian process regressions work well for smooth functions, assuming that two close points are unlikely to yield two very different values, but this is not the case with the daily realized volatility, where large spikes are commonly observed in our dataset. In addition to this, the training time is a real issue known within the data science community but surprisingly not well documented. One of the Machine Learning Coursera courses from UCSanDiego HSE suggested to replace the training dataset with a small number of points, using a sparse G.P. approach for example or a method similar to the one used in Support Vector Machine, but those are options we ran out of time to implement.

#### 3.2.2.2 XGBoost

Lastly we model the relationship between the averaged lagged realized volatilities and the future one using XGBoost. The process is similar to the one used for G.P.R., except that we do not need to specify a kernel shape (the one used in the Bayesian Optimization part is the R.B.F. by default) but we have another set of hyperparameters to improve. Among them, the most important ones are the minimum loss required to make a further partition on a leaf node of the tree $\gamma$, the learning rate $\eta$ and the maximum depth of a tree $max_depth$.
First of all, the fastness of XGBoost is absolutely impressive; it took much less time to fit the 15 models with it than to fit one model with G.P.R., no matter the optimization technique used. In the same manner, we only display the predictions for the last year and a half to better see the differences in predictions depending on the tuning method. We obtained:

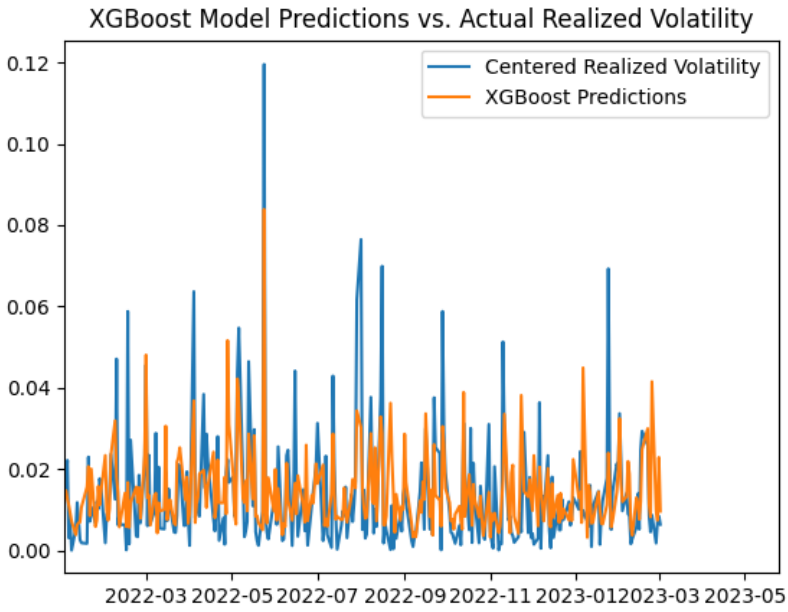


Figure 3.4: XGBoost predictions on a subset, Grid search.

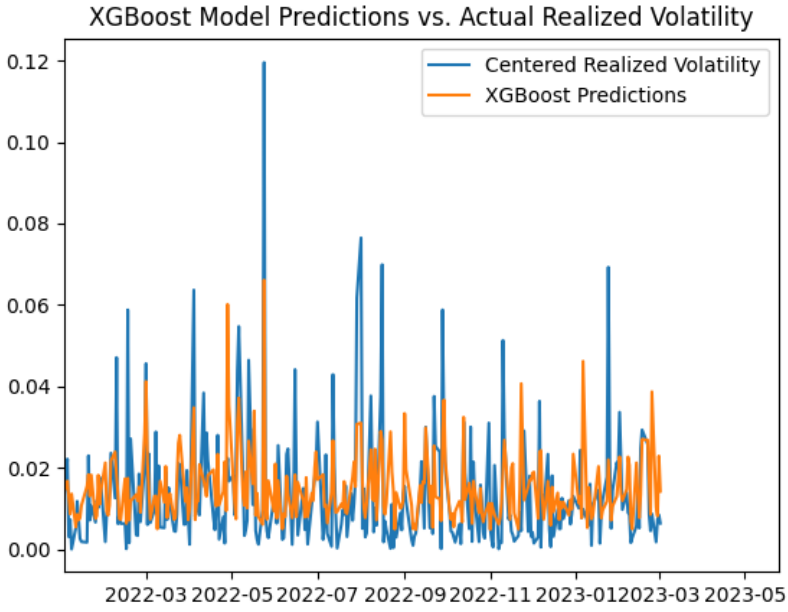


Figure 3.5: XGBoost predictions on a subset, Random search.

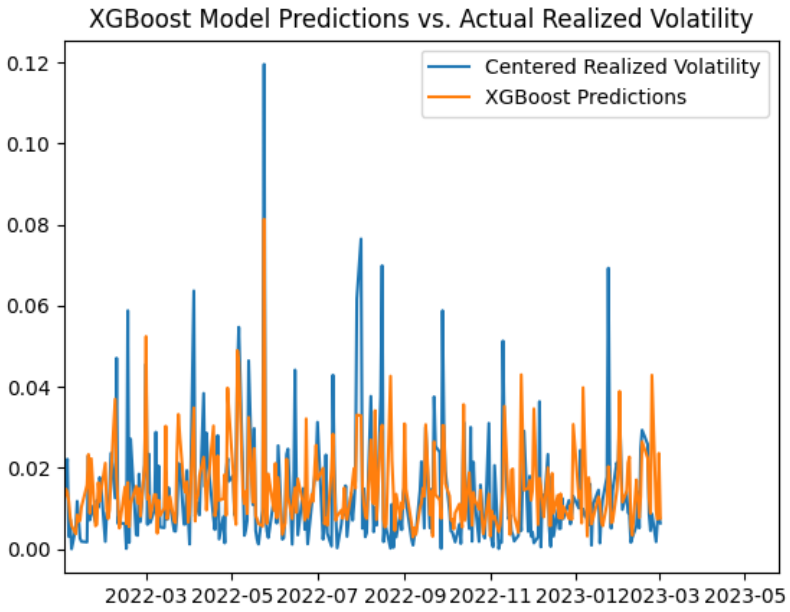


Figure 3.6: XGBoost predictions on a subset, Bayesian optimization.

The main observation here is that the values taken by the model tuned with random search are less volatile and more likely to fall within a narrow range compared to the two other models. It is striking when looking at the major spike of June 2022 but for also for every day where the volatility is close to 0, where the predictions made by the models tend to overshoot the actual R.V. We can therefore expect this model to be less accurate than its peers, which is confirmed by the following table:

| Mean MAE Grid search | Mean MSE Grid search | Mean MAE Random search | Mean MSE Random search | Mean MAE Bayesian optimization | Mean MSE Bayesian optimization |
|---|---|---|---|---|---|
| 0.00703975 | 0.000121380 | 0.00911824 | 0.000184236 | 0.00751451 | 0.000134258 |

Table 3.5: Averaged MAE & MSE for XGBoost across the universe of stocks.

The best performing model on average is therefore the one tuned with the Grid search method, and is also the one which took the longest to train. To understand this, let us look at the average values of the hyperparameters between the three models:

| | colsample bytree | gamma | learning rate | max depth | min child weight | n estimators | subsample |
|---|---|---|---|---|---|---|---|
| Bayesian optim. | 0.93 | 0.60 | 0.17 | 6.53 | 5.53 | 95.87 | 0.67 |
| Grid search | 1 | 0 | 0.1 | 3 | 7.40 | 100 | 0.77 |

Table 3.6: Hyperparameters by Grid search and Bayesian optimization.

As only two out of the fifteen models tuned with grid search had a gamma different of 0 (see Appendix B), it is likely that these discrepancies in performance come from the maximum depth of the regression trees and/or the minimum weight of the children. Both these hyperparameters control how the models build its trees, with the minimum child weight being the minimum

required value of the objective function to make a split and the maximum depth controlling the complexity of the tree. It is important to note that there is a strong risk of overfitting by picking "high" values of depth for a tree since it tends to fit the noise in the training data, rather than the underlying patterns.

We also look at which features are the most important to the model by focusing on the Apple stock for both Grid search and B.O. Without going into the mathematical details of how the importance of a feature is computed in the `sklearn` package, it reflects the improvement in accuracy brought by a feature to the branches it is on in the XGBoost regression trees. We fitted an XGBoost model for both sets and obtained the following graphs:

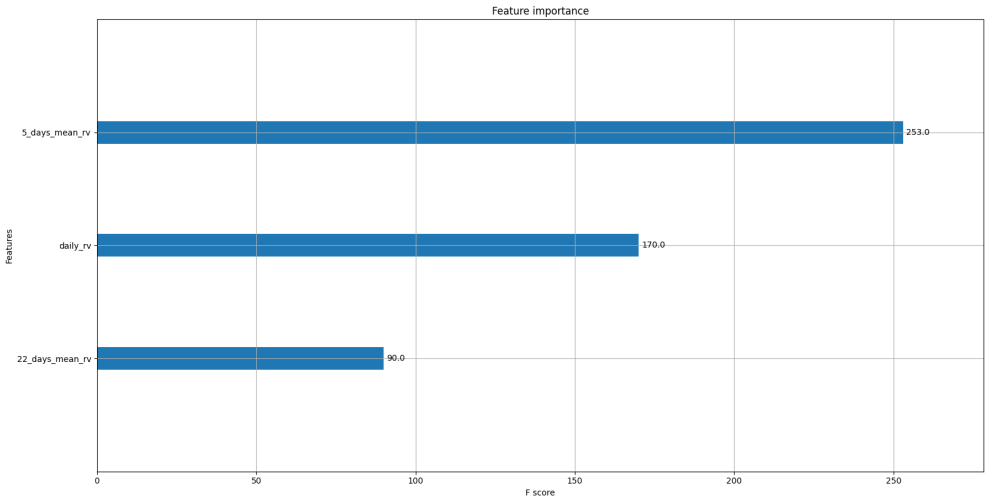


Figure 3.7: Importance of features for the XGBoost model tuned with Grid search.

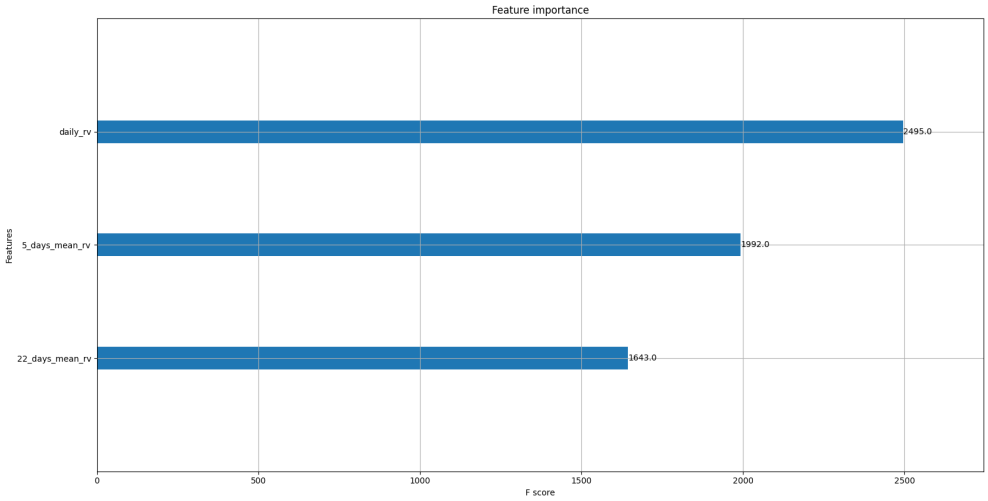


Figure 3.8: Importance of features for the XGBoost model tuned with Bayesian optimization.

Despite the two models not agreeing on which feature is the most relevant, there is however a consensus on the fact that the 22 days averaged R.V. is indeed less important, and by far

for the first model. This makes sense on the financial side knowing that the two other factors incorporate more recent information which tends to matter most for participants in the market. This is however not a factor to neglect since it still possesses a relatively high explanatory power and overall helps assess the longer trends in the market as well as some potential seasonality effects.

Overall when we compare the results from the best linear model (W.L.S.) with those from the XGBoost model tuned with Grid search with the table 3.2 and 3.5, it appears that the linear H.A.R. has a higher averaged Mean Absolute Error but a smaller Mean Squared Error metric. One of the reasons for these similarities in the performances is that machine learning algorithms handle very well high frequency data and make a real difference in these situations, while the traditional H.A.R. model is known to perform extremely well with higher time-frames data. Although there is no absolute winner based solely on these two metrics, both sets of models agree on the fact that the daily lagged realized volatility and the five days one are the most important explanatory variables in the model.

# Chapter 4

# Conclusion

The purpose of this Master's Thesis was to investigate various models used for predicting realized volatility one day ahead for several stocks and identify the most important features among those commonly used in the classic heterogeneous autoregressive model.
In the first chapter, we presented some empirical facts about realized volatility and in the following two chapters, we developed, described, and compared the different models. However, we regret not exploring higher time-frame forecasts or other problem specifications in the Gaussian Process Regression part of the study.

To further develop this study, which proves itself to be very actual given the numerous recent papers and the surge of Machine Learning in the financial industry recently, several leads could be explored. First of all, the use of high-frequency data would be extremely interesting to exploit, as well as other explanatory variables such as those detailed in the first Chapter. In addition to this, splitting the universe of stocks by sector for example instead of market capitalization could show great benefits towards more accurate models and it is economically sound to think that stocks within the same industry might behave more closely than stocks with the same market capitalization. Regarding the way of building the models, other kernels could be tested, even linear combinations of those employed in this study, as well as other hyperparameters tuning methods such as genetic algorithms or manually narrowing the grid search region iteration after iteration. It should be noted that these techniques are extremely expensive in terms of computational resources and might not be suited for everyone. I would like to conclude this master's thesis with the following quote from Francois Chollet, author of the book Deep Learning with Python and engineer at Google: *"Training a machine learning model is like trying to find a needle in a haystack, except the needle is also made of hay."*

# Chapter 5

# Bibliography


Abourachid et al., 2017. Momentum strategies in European equity markets: Perspectives on the recent financial and European debt crises. *Finance Research Letters*, Volume 23, pp. 147-151.

Adams et al., 2012. Practical Bayesian Optimization of Machine Learning Algorithms.

Ahmad, R., Wilmott, P., 2005. Which Free Lunch Would You Like Today, Sir?: Delta Hedging, Volatility Arbitrage and Optimal Portfolios.

Bollerslev T., 1986. Generalized Auto Regressive Conditional Heteroskedasticity. *Journal of Econometrics*, Volume 31, pp. 307–327.

Bollerslev et al., 2015. The VIX, the Variance Premium and Stock Market Volatility.
Brahim-Belhouari S., Bermak A., 2004. Gaussian process for nonstationary time series prediction. *Computational Statistics & Data Analysis*, Volume 47, pp. 705–712.

Brownlee J., 2016. Gradient Boosted Trees With XGBoost and scikit-learn.

Campbell Y. and Shiller R., 1988. The Equity Premium and the Volatility Spread.
Carr et al., 2020. Using Machine Learning to predict realized variance. *Journal of Investment Management*, Volume 18, pp. 1-16.

Chen T., Guestrin C., 2016. XGBoost: A Scalable Tree Boosting System.

Christensen K., Podolskij M., 2006. Realized range-based estimation of integrated variance. *Journal of Econometrics*, Volume 141, pp. 323–349.

Clements A., Preve D., 2019. A Practical Guide to Harnessing the HAR Volatility Model.

Corsi F., 2003. A Simple Approximate Long-Memory Model of Realized Volatility. *Journal of Financial Econometrics*, Volume 7, pp. 174–196.

Engle R., 1982. Autoregressive conditional heteroskedasticity with estimates of variance of UK inflation. *Econometrica*, Volume 50, pp. 987–1008.

Frazie P., 2018. A Tutorial on Bayesian Optimization.

Frias R., 2012. Gaussian Processes for regression: a tutorial.

Gatheral J., 2006. The volatility surface: A Practitioner's Guide.

Ghahramani Z., 2011. A Tutorial on Gaussian Processes, Class handout, Cambridge University.

Han et al., 2016. Gaussian Process Regression Stochastic Volatility Model for Financial Time Series. *Journal of selected topics in signal processing*, Volume 10, pp. 1015-1028.

Joy et al., 2016. Hyperparameter Tuning for Big Data using Bayesian Optimisation.

Kambouroudis et al., 2021. Forecasting realized volatility: The role of implied volatility, leverage effect, overnight returns, and volatility of realized volatility. *Journal of Futures Market*, Volume 41, pp. 1618-1639.

Karasan, A., 2021. Machine Learning for Financial Risk Management with Python.

Klein A., Falkner S., 2017. Fast Bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics*, Volume 11, pp. 4945–4968.

Kumbure et al., 2022. Machine learning techniques and data for stock market forecasting: A literature review.

Liu et al., 2020. An Overview of Gaussian process Regression for Volatility Forecasting.

Lomas R., 2005. How to estimate the volatility of an asset with high-frequency data, working paper.

Rasmussen C., Williams C., 2006. Gaussian Processes for Machine Learning. *The MIT Press.*

Roberts et al., 2013. Gaussian processes for time-series modelling. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, Volume 371, pp. 1-25.

Romano S., 2019. Machine Learning for volatility forecasting.

Sander R., 2021. Gaussian Process Regression From First Principles.

Teller et al., 2022. Short- to Long-Term Realized Volatility Forecasting using Extreme Gradient Boosting.

Torben Andersen, G., Bollerslev, T., Diebold, X., Labys, P., 1986. Modeling and forecasting realized volatility.

Qiu et al., 2019. Versatile HAR model for realized volatility: A least square model averaging perspective. *Journal of Management Science and Engineering*, Volume 4, pp. 55-73.

Wu et al., 2014. Gaussian Process Volatility Model.

Zhang et al., 2022. Volatility forecasting with machine learning and intraday commonality.

# Chapter 6

# Appendices

## 6.1 Appendix A

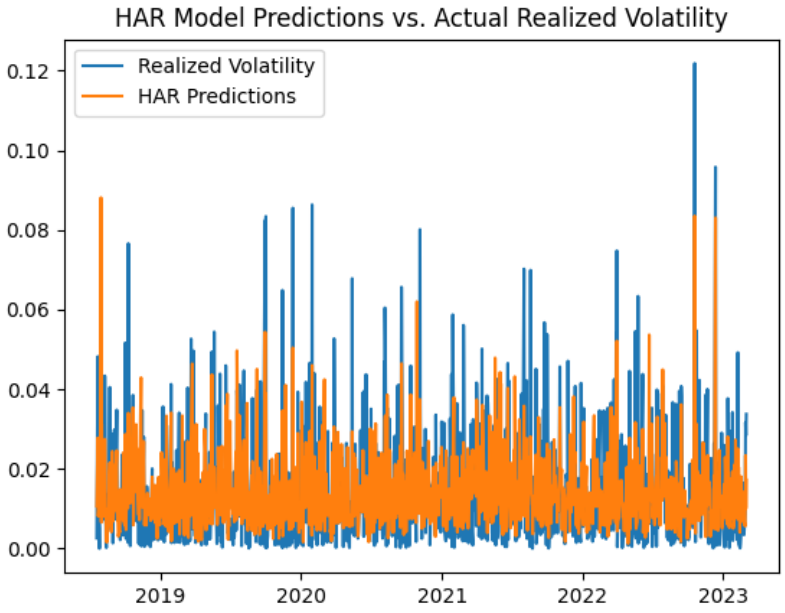


Figure 6.1: Whole dataset predictions for the H.A.R. model estimated with the O.L.S.

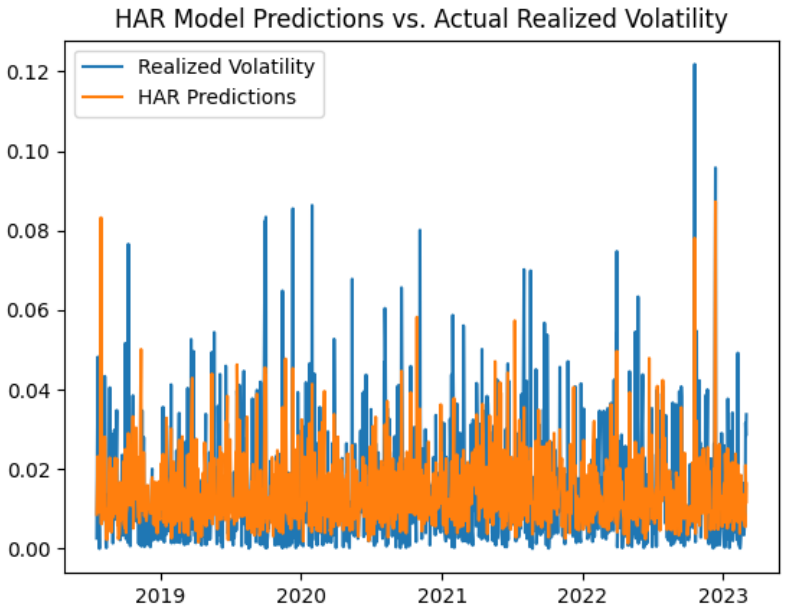


Figure 6.2: Whole dataset predictions for the H.A.R. model estimated with the N.N.L.S.

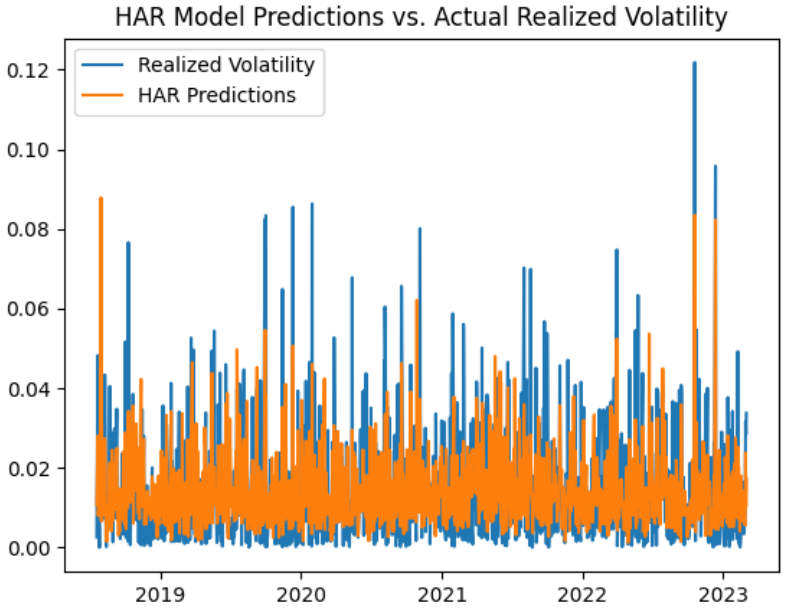


Figure 6.3: Whole dataset predictions for the H.A.R. model estimated with the W.L.S.

| Symbol | MSE OLS | MAE OLS | MSE NNLS | MAE NNLS | MSE WLS | MAE WLS |
|---|---|---|---|---|---|---|
| AAPL | 0.000112504 | 0.00766737 | 0.000119348 | 0.00787348 | 0.000112396 | 0.00766575 |
| ADBE | 0.000153764 | 0.00813997 | 0.000152246 | 0.00823961 | 0.000153665 | 0.00815990 |
| AMD | 0.000300892 | 0.0125919 | 0.000316165 | 0.0129938 | 0.000300729 | 0.0125894 |
| AMZN | 0.000170241 | 0.00902728 | 0.000182818 | 0.00941303 | 0.000169992 | 0.00901630 |
| BA | 0.0000919220 | 0.00676665 | 0.0000941279 | 0.00690654 | 0.0000918331 | 0.00676324 |
| BAC | 0.000134733 | 0.00690751 | 0.000136473 | 0.00709025 | 0.000135142 | 0.00683830 |
| COST | 0.0000666517 | 0.00553798 | 0.0000695124 | 0.00566777 | 0.0000661757 | 0.005517508 |
| HD | 0.0000710322 | 0.00585529 | 0.0000764930 | 0.00609113 | 0.0000709953 | 0.00585404 |
| INTC | 0.0000963698 | 0.00678021 | 0.0000990585 | 0.00699458 | 0.0000961600 | 0.00677536 |
| JNJ | 0.0000298439 | 0.00380739 | 0.0000313087 | 0.00391301 | 0.0000297628 | 0.00380800 |
| JPM | 0.0000933247 | 0.00644154 | 0.000103968 | 0.00675829 | 0.0000918276 | 0.00639311 |
| MSFT | 0.0000602700 | 0.00546162 | 0.0000648417 | 0.00568829 | 0.0000601059 | 0.00545283 |
| NEE | 0.0000441714 | 0.00480973 | 0.0000470448 | 0.00500171 | 0.0000441950 | 0.00480905 |
| NVDA | 0.000258559 | 0.0109864 | 0.000270656 | 0.0113860 | 0.000257003 | 0.0109573 |
| XOM | 0.0000477282 | 0.00499239 | 0.0000519312 | 0.00519499 | 0.0000478201 | 0.00500304 |

Table 6.1: MSE and MAE for each stock and each regression type.

| Symbol | constant | p-value | daily rv | p-value | 5 days mean rv | p-value | 22 days mean rv | p-value |
|---|---|---|---|---|---|---|---|---|
| AAPL | 0.0002 | 0.608 | -0.2134 | 0.000 | 1.2557 | 0.000 | -0.0593 | 0.08 |
| ADBE | -0.0001 | 0.705 | -0.2173 | 0.000 | 1.2759 | 0.000 | -0.0482 | 0.169 |
| AMD | 0.0006 | 0.383 | -0.2338 | 0.000 | 1.2330 | 0.000 | -0.0295 | 0.443 |
| AMZN | 0.0000 | 0.999 | -0.244 | 0.000 | 0.2839 | 0.000 | -0.0431 | 0.213 |
| BA | 0.0002 | 0.519 | -0.2369 | 0.000 | 1.2727 | 0.000 | -0.0547 | 0.114 |
| BAC | 0.0003 | 0.304 | -0.1377 | 0.000 | 1.2252 | 0.000 | -0.1084 | 0.000 |
| COST | 0.000 | 0.685 | -0.1482 | 0.000 | 1.1876 | 0.000 | -0.0620 | 0.0740 |
| HD | 0.0001 | 0.635 | -0.2289 | 0.000 | 1.2177 | 0.000 | -0.0055 | 0.875 |
| INTC | 0.000 | 0.987 | -0.2620 | 0.000 | 1.2530 | 0.000 | 0.0067 | 0.849 |
| JNJ | 0.000 | 0.613 | -0.2205 | 0.000 | 1.2773 | 0.000 | -0.0719 | 0.038 |
| JPM | 0.000 | 0.866 | -0.1930 | 0.000 | 0.1944 | 0.000 | -0.0081 | 0.792 |
| MSFT | 0.000 | 0.916 | -0.206 | 0.000 | 1.1977 | 0.000 | 0.0063 | 0.859 |
| NEE | 0.0001 | 0.507 | -0.2173 | 0.000 | 1.2577 | 0.000 | -0.0485 | 0.149 |
| NVDA | 0.000 | 0.885 | -0.2086 | 0.000 | 1.2972 | 0.000 | -0.0934 | 0.004 |
| XOM | 0.000 | 0.807 | -0.2368 | 0.000 | 1.2478 | 0.000 | -0.0203 | 0.546 |

Table 6.2: Coefficients of the H.A.R. model estimated with the O.L.S. and p-values associated.

| Symbol | constant | p-value | daily rv | p-value | 5 days mean rv | p-value | 22 days mean rv | p-value |
|---|---|---|---|---|---|---|---|---|
| AAPL | 0.0001 | 0.635 | -0.2230 | 0.000 | 1.2490 | 0.000 | -0.0382 | 0.157 |
| ADBE | -0.000 | 0.862 | -0.2408 | 0.000 | 1.2226 | 0.000 | 0.0210 | 0.399 |
| AMD | 0.0001 | 0.795 | -0.2229 | 0.000 | 1.2220 | 0.000 | -0.0089 | 0.745 |
| AMZN | 0.000 | 0.830 | -0.2574 | 0.000 | 1.2714 | 0.000 | -0.0205 | 0.424 |
| BA | 0.000 | 0.967 | -0.2322 | 0.000 | 1.2488 | 0.000 | -0.0193 | 0.460 |
| BAC | 0.000 | 0.561 | -0.2277 | 0.000 | 1.2690 | 0.000 | -0.0465 | 0.000 |
| COST | 0.000 | 0.611 | -0.2114 | 0.000 | 1.2216 | 0.000 | -0.0316 | 0.246 |
| HD | 0.0001 | 0.745 | -0.2184 | 0.000 | 1.2261 | 0.000 | -0.0177 | 0.493 |
| INTC | 0.000 | 0.632 | -0.2572 | 0.000 | 1.2657 | 0.000 | -0.0185 | 0.522 |
| JNJ | 0.000 | 0.811 | -0.2432 | 0.000 | 1.2568 | 0.000 | -0.0201 | 0.435 |
| JPM | 0.000 | 0.882 | -0.2508 | 0.000 | 1.2596 | 0.000 | -0.0106 | 0.644 |
| MSFT | 0.000 | 0.993 | -0.2335 | 0.000 | 1.2176 | 0.000 | 0.0113 | 0.676 |
| NEE | 0.000 | 0.996 | -0.2383 | 0.000 | 1.2571 | 0.000 | -0.0107 | 0.680 |
| NVDA | 0.000” | 0.328 | -0.2359 | 0.000 | 1.2631 | 0.000 | -0.0447 | 0.094 |
| XOM | 0.000 | 0.849 | -0.2346 | 0.000 | 1.2083 | 0.000 | 0.0246 | 0.330 |

Table 6.3: Coefficients of the H.A.R. model estimated with the W.L.S. and p-values associated.

## 6.2 Appendix B

### 6.2.1 Gaussian Process Regression

### 6.2.2 XGBoost

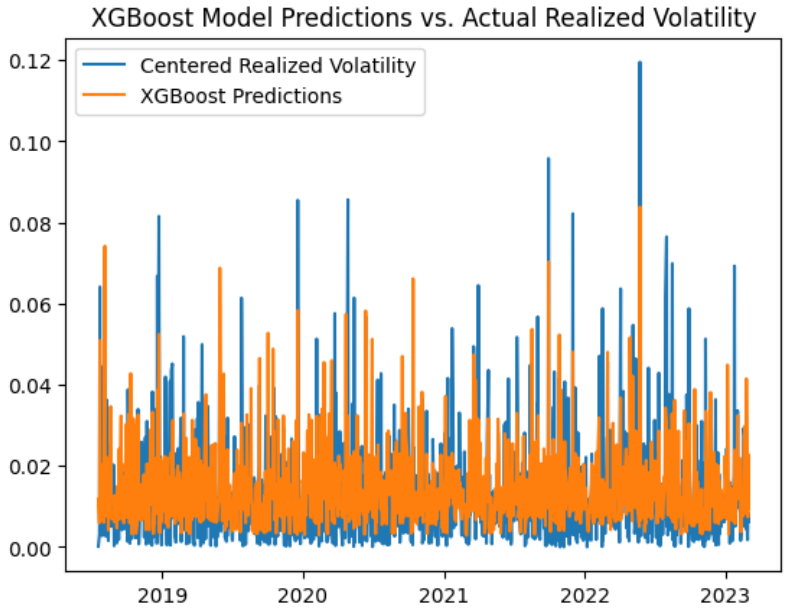


Figure 6.4: Whole dataset predictions for the XGBoost model estimated with Grid Search.

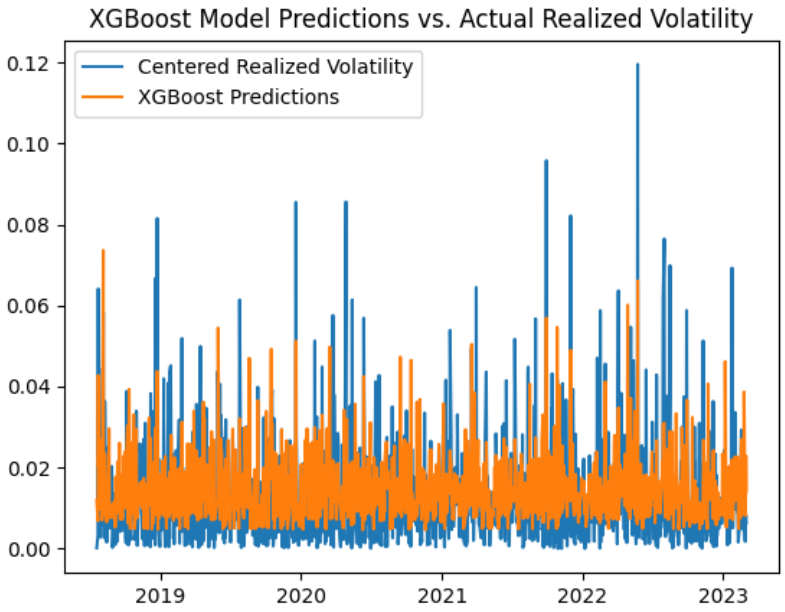


Figure 6.5: Whole dataset predictions for the XGBoost model estimated with Random Search.

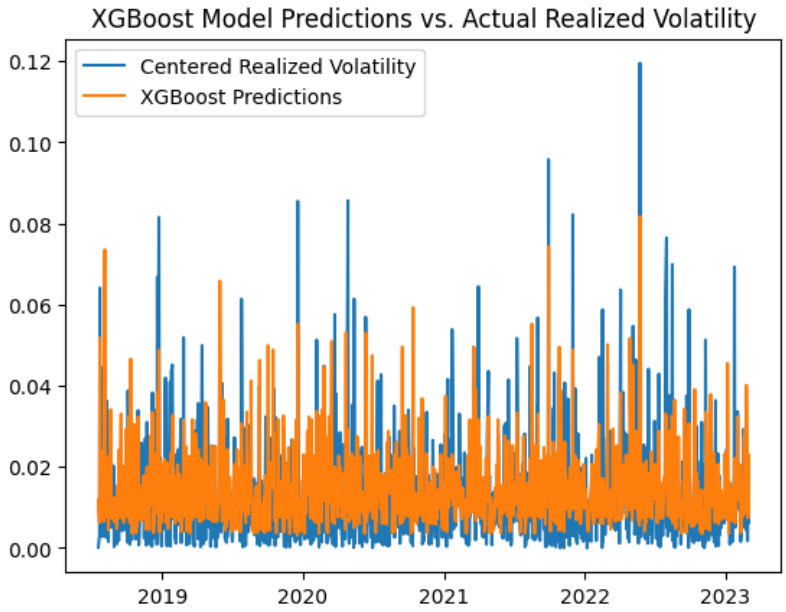


Figure 6.6: Whole dataset predictions for the XGBoost model estimated with Bayesian Optimization.

| Symbol | MSE Grid search | MAE Grid search | MSE Random search | MAE Random search | MSE Bayesian Optimization | MAE Bayesian Optimization |
|---|---|---|---|---|---|---|
| AAPL | 0.000112082 | 0.00746609 | 0.000179154 | 0.00985302 | 0.000112162 | 0.00752273 |
| ADBE | 0.000153407 | 0.00807254 | 0.000238777 | 0.0106154 | 0.000144386 | 0.00810564 |
| AMD | 0.0003210170 | 0.0128435 | 0.000419450 | 0.0150130 | 0.000320807 | 0.0129070 |
| AMZN | 0.0002071630 | 0.00910977 | 0.000293560 | 0.0114454 | 0.000234153 | 0.00944328 |
| BA | 0.0000934942 | 0.00673216 | 0.000143660 | 0.00855005 | 0.0000897672 | 0.00669349 |
| BAC | 0.000126074 | 0.00677336 | 0.000189284 | 0.00908435 | 0.000126538 | 0.00689599 |
| COST | 0.0000618316 | 0.00541984 | 0.0001003222 | 0.00718500 | 0.0000666108 | 0.00556728 |
| HD | 0.0000710218 | 0.00584441 | 0.000119881 | 0.00795225 | 0.0000872670 | 0.00636975 |
| INTC | 0.0000818787 | 0.00650473 | 0.000143506 | 0.00883117 | 0.000143587 | 0.00884463 |
| JNJ | 0.0000272712 | 0.00368217 | 0.0000428549 | 0.00480946 | 0.0000342215 | 0.00401136 |
| JPM | 0.000107684 | 0.00651436 | 0.000183122 | 0.00878168 | 0.000203100 | 0.00916650 |
| MSFT | 0.0000646922 | 0.00561675 | 0.000105733 | 0.00752733 | 0.0000646379 | 0.00565621 |
| NEE | 0.0000467217 | 0.00469178 | 0.0000905385 | 0.00636010 | 0.0000491188 | 0.00492014 |
| NVDA | 0.000293506 | 0.01136030 | 0.0004211752 | 0.01416811 | 0.000280213 | 0.01144717 |
| XOM | 0.0000528619 | 0.00496447 | 0.0000888708 | 0.00659799 | 0.0000573043 | 0.00516645 |

Table 6.4: MSE and MAE for each stock and method of hyperparameter tuning.

| Symbol | colsample bytree | gamma | learning rate | max depth | min child weight | n estimators | subsample |
|---|---|---|---|---|---|---|---|
| AAPL | 1.0 | 0 | 0.25 | 10 | 1 | 50 | 0.568 |
| ADBE | 0.845 | 0 | 0.0554 | 4 | 1 | 187 | 0.559 |
| AMD | 1.0 | 0 | 0.113 | 3 | 10 | 200 | 0.769 |
| AMZN | 1.0 | 0 | 0.25 | 10 | 1 | 50 | 0.719 |
| BA | 1.0 | 0 | 0.25 | 10 | 1 | 50 | 0.574 |
| BAC | 1.0 | 0 | 0.25 | 10 | 1 | 50 | 0.574 |
| COST | 1.0 | 0 | 0.145 | 3 | 10 | 50 | 0.5 |
| HD | 0.887 | 9 | 0.118 | 6 | 6 | 151 | 0.677 |
| INTC | 0.714 | 0 | 0.127 | 10 | 10 | 50 | 1.0 |
| JNJ | 1.0 | 0 | 0.127 | 3 | 10 | 50 | 0.5 |
| JPM | 0.500 | 0 | 0.0763 | 3 | 10 | 200 | 0.999 |
| MSFT | 1.0 | 0 | 0.25 | 10 | 1 | 50 | 0.574 |
| NEE | 1.0 | 0 | 0.145 | 3 | 10 | 50 | 1.0 |
| NVDA | 1.0 | 0 | 0.25 | 10 | 1 | 50 | 0.574 |
| XOM | 1.0 | 0 | 0.142 | 3 | 10 | 200 | 0.5 |

Table 6.5: Set of hyperparameters estimated with Grid search.