APARCH Models Estimated by Support Vector Regression

Applied to Financial Volatility Estimation

Arne Ladstein Waagbø

A thesis presented for the degree of
Master of Statistics
Financial Theory and Insurance Mathematics

UNIVERSITY OF BERGEN

Faculty of Mathematics and Natural Sciences
01.06.2021

Acknowledgements

I would very much like to thank my supervisor Yushi Li for her guidance and help in the preparation of this thesis. I am gratefull for the engaging and dedicated teaching and support staff I have met at UiB during my studies. Finally, I am thankfull for my great friends and family for their continuos support and inspiration.

Abstract

This thesis presents a comprehensive study of asymmetric power autoregressive conditional heteroschedasticity (APARCH) models for modelling volatility in financial return data. The goal is to estimate and forecast volatility in financial data with excess kurtosis, volatility clustering and asymmetric distribution. Models based on maximum likelihood estimation (MLE) will be compared to the kernel based support vector regression (SVR). The popular Gaussian kernel and a wavelet based kernel will be used for the SVR. The methods will be tested on empirical data, including stock index prices, credit spreads and electric power prices. The results indicate that asymmetric power models are needed to capture the asseymtry in the data. Furthermore, SVR models are able to improve estimation and forecasting accuracy, compared with the APARCH models based on MLE.

Contents

List of Figures

List of Tables

1 Introduction

Traditional economic models such as capital asset pricing model (CAPM) (Treynor 1961,1962; Sharpe 1964; Lintner 1965; Mossin 1966), based upon the modern portfolio theory of Markowitz (1952), are based upon the variance and correlations of financial assets. The expected return of an asset is dependent on its correlation to the market return, and its volatility compared to the market volatility. The problem is that both the volatility of markets and of individual securities varies over time, and so does the correlation between them. Volatility is also instrumental to the pricing of financial options contracts, and it is in fact the only unobservable parameter in the famous Black-Scholes option pricing formula (Black, Scholes 1973). With known market prices of options, the problem can be inverted to calculate implied volatility. Knowledge about the volatility of asset prices is crucial for financial risk management, and without it there would be no risk. Thus, models that can reliably estimate and forecast volatility is of great importance to speculators, investors, industrial purchasers and hedgers and other market participants.

Financial time series often possesses characteristics such as heavy tails, volatility clusters, asymmetric leverage effects and dependence without correlation, the so-called stylized features of financial time series. Models based on the ARCH framework of Engle (1982) and the GARCH framework of Bollerslev (1986) are popular for their ability to capture volatility clusters often observed in financial data. These models are often fitted with assumption of gaussian error terms, but other distributions are also supported to accommodate fat tails. Asymmetric leverage effects describe the markets tendency to experience increased volatility with bad news, and lowered volatility with good news (Black 1976). To support asymmetry, Ding et al. (1993), introduced the APARCH models. The APARCH model has two extra parameters compared to the GARCH model, which allows for greater flexibility. With

this model, conditional volatility will increase as financial asset prices falls and decrease with rising prices (for most assets, the opposite is also possible). An explanation for the leverage effect is that as equity prices declines, so does the equity to debt ratio. This increases the risk of the debt, and so it increases the volatility of the equity. This also works in reverse as increased volatility also increases the required expected return of the asset, and thus a lower price is required. Nelson (1991), Glosten, Jaganathan and Runkle (1989) and Engle and Ng (1992) showed the importance of including asymmetry in financial time series models. There is usually little correlation in the daily returns of financial assets, otherwise there would be inconsistencies with the efficient market hypothesis. There is evidence of a slight negative correlation in the second lag, meaning that there is a mean reversion effect (Ding et al. 1993). This mean reversion effect is strongest after extreme moves in either direction and suggests that the return series is not independent and identically distributed (iid). Furthermore, the absolute return or squared return often contains substantial autocorrelation even for long lags (Taylor 1986). This is clear evidence that return series are not iid, and that GARCH type models are necessary. Ding et al. showed that the power transformation of the return series for the S&P 500 index, $|r_t|^d$, had significant autocorrelations for all lags up to 100, for all $d \in (0.25, 3)$. The strongest autocorrelations were found for values of d close to one. Ding et al. further showed that the significant positive autocorrelation lasted for more than 2500 trading days, more than 10 years, for the S&P 500. This is called the long memory property of financial returns.

These models are usually estimated by maximum likelihood estimation (MLE), but newer research has been done with estimation based on support vector regression (SVR). As the MLE need Gaussian distributed residuals to be efficient, non-parametric models like the SVR can be helpful in dealing with financial data. Support vector regression is an extension to the support vector machines that were introduced by Vladimir Vapnik and colleagues

at AT&T Bell Laboratories (now Nokia Bell Labs) in the 1990s, (Boser et al. 1992; Cortez and Vapnik 1995). The SVM is a generalization of the Generalized Portrait algorithm (Vapnik and Lerner 1963; Vapnik and Chervonenkis 1964). The support vector machine is a powerful method for binary classification and can efficiently perform a non-linear classification using the so-called kernel trick. The kernel trick involves implicitly mapping inputs into high-dimensional feature spaces, where the mapping can be achieved by utilization of kernel functions. SVMs have strong rooting in optimization theory, also called Vapnik-Chervonenkis (VC) theory, developed by Vapnik (Vapnik and Chervonenkis 1974; Vapnik 1982; Vapnik 1995). VC theory characterizes properties of learning machines which enable them to generalize well to unseen data, their rate of convergence and conditions for consistency. SVM was initially applied to optical character recognition and has since found many more applications. They have many desirable features, including strong theoretical mathematical support, flexibility with robustness to overfitting, unique solutions and good empirical performance. Support vector regression is entirely data driven and needs no assumption about the underlying distribution of the dataset. The choice of kernel in the non-linear mapping is very important. This thesis will compare the performance of support vector regression with the popular Gaussian kernel and the more recent wavelet kernel proposed by Zhang *et al* (2004) to models estimated by MLE. The wavelet kernel showed improved forecasting performance applied to simulation data in Li (2014) and applied to oil price volatility in Li and Karlsson (2020).

Other previous studies of volatility estimation using SVM are summarized in the following paragraph. Chen *et al* (2010) found that SVM-GARCH, with a Gaussian kernel, performed better than SMV-GARCH with linear and polynomial kernel, as well as MLE GARCH, EGARCH (Nelson 1991) and ANN-GARCH in Monte Carlo simulation as well as real data (GBP/USD exchange rate and NYSE composite index). They noted that in the presence

of strong asymmetry the EGARCH performed well. Pérez-Cruz *et al* (2003) compared the performance of GARCH(1,1) models estimated by maximum likelihood and SVR (unspecified kernel) applied to forecasting stock market (S&P100, FTSE100, IBEX35 and NIKKEI) indices and single stock (GM and HP) return volatility. They compared the r^2 in- and out of sample. The out of sample results were better performance of the SVR-GARCH in all but one instance. They attribute the improved performance of the SVR-GARCH to not trying to fit Gaussian distributed residuals. Ou and Wang (2010) compared semi-parametric method, LSSVM (Least square support vector machine) by Suykens et al. (1999), with the classical GARCH(1,1), EGARCH(1,1) and GJR(1,1) models to forecast financial volatilities of three major ASEAN stock markets. Their experimental results suggest that using hybrid models, GARCH-LSSVM, EGARCH-LSSVM and GJR-LSSVM provides improved performances in forecasting the leverage effect volatilities. Bildirici & Ersin (2009) fitted neural networks based on nine different models of GARCH family, to forecast Istanbul stock volatility and most of the hybrid models improved forecasting performance. Neural network models have a major drawback compared to SVM models as they are prone to get stuck at local optimums. Bezerra *et al* (2017) used a linear combination of one, two, three and four Gaussian kernels in the SVR based on GARCH(1,1) to take into account the existence of market regimes. Nikkei 225 and Ibovespa daily returns were used as the dataset. The empirical results indicate that the mixture of Gaussian kernels can improve the SVR–GARCH one-period-ahead volatility forecasts. The best performance was the SVR–GARCH with a mixture of three Gaussian kernels. The SVR–GARCH with a mixture of four Gaussian kernels and the SVR-GARCH with a wavelet kernel also performed well. All SVR–GARCH models significantly outperformed every MLE-GARCH model, including GJR and EGARCH. Peng *et al* (2018) evaluated SVR-GARCH's predictive performance of daily and hourly volatility of three cryptocurrencies (Bitcoin, Etherum and Dash) and three exchange rate pairs (JPY, GBP and Euro), all

in relation to USD. The results showed that SVR-GARCH models managed to outperform all nine GARCH benchmarks – GARCHs, EGARCHs and GJR-GARCHs with Normal, Student's t and Skewed Student's t distributions. Sun and Yu (2020) propose a two-stage forecasting volatility method by combining the SVR and GARCH models, where they first find a volatility estimate by MLE GARCH which is then used as input to a SVR. They used the S&P 500 index and the GBP/USD exchange rate in empirical analysis and found that their hybrid model improved the volatility forecasting ability compared to both MLE-GARCH and traditional SVR-GARCH.

This thesis is organized as follows. Chapter 2 will give a brief overview of time series models that are commonly used for financial data. We then introduce the APARCH model, which have some great features for financial volatility estimation. Chapter 3 gives the theoretical background of the Support Vector Machines used for classification. We present the kernel trick, which makes the SVMs able to efficiently solve non-linear classification problems. We then make the necessary extensions so that we can use the method of support vectors for regression problems. Section 4 shows how we can use Support Vector regression to estimate APARCH models. We also present a wavelet based kernel that we will use and compare to the Gaussian kernel in Section 5. Finally, section 5 applies APARCH models estimated by MLE and by SVR to financial volatility estimation and forecasting in three different datasets. Chapter 6 gives a summary of the thesis and the empirical results from chapter 5.

2 APARCH models

Financial data is often recorded and presented in the form of time series. Financial return series are an example of this. One obvious property of financial return series is that volatility is not independent of time. To better capture time varying volatility Robert Engle (1982) introduced the autoregressive conditional heteroscedasticity (ARCH) model. Since then, extensions to the ARCH model has been made to incorporate other typical features of financial return series such as heavy tails, asymmetry and long memory. The asymmetric power ARCH model introduced by Ding, Granger and Engle (1993) is a poplar such extension. This chapter will give a short introduction to time series and time series models, leading up to the APARCH model.

2.1 Time series models

A time series is a set of data points ordered by the time of their observation. The times of the observations are not necessarily equidistant, but are commonly sampled as daily, monthly, yearly etc. statistics. Time series have applications in a number of fields and are very important in finance. Examples includes daily prices of financial assets, quarterly earnings numbers for public corporations and annual GDP data. The objectives of analyzing time series is to draw some inferences about the data. We might observe trends, seasonality and variability or other statistics of the data. Filtering of the data can separate noise from signal. We can observe how a time series relates to another, for example how corporate earnings relate to GDP numbers. Time series can be used for prediction, as we can observe how data points relate to previous data and then project from the most recent data. If we specify a model for a time series we can simulate it using statistical software.

We let x_t denote an observation of some data point at time t, and let $\boldsymbol{x_t}$ be a set of such observations. The observation x_t is supposed to be a realization

of the random variable X_t.

We can now make some definitions. The definition of a time series model is a specification of the joint distribution of random variables X_t indexed by time order. x_t is a realization of the random variable sequence X_t.

Let X_t be a time series with finite first- and second-order moments, ie. $E[x_t^2] < \infty$. Then the mean function of x_t is defined as

$$\mu_X(t) = E[X_t]$$

The covariance function of x_t is defined as

$$\gamma_X(s,t) = Cov(X_s, X_t) = E[(X_s - E[X_s])(X_t - E[X_t])], \quad \forall \text{ integers } s,t$$

x_t is a weakly stationary time series if for all t

$$\mu_X(t) \text{ is independent of } t$$

and for all h

$$\gamma_X(t, t+h) \text{ is independent of } t$$

If additionally

$$(X_1, ..., X_n) \stackrel{d}{=} (X_{1+h}, ..., X_{n+h}), \quad \forall \text{ integers } h \text{ and } n \geq 1$$

then X_t is a strictly stationary time series. If X_t is strictly stationary, then it is also weakly stationary.

For a stationary time series we notice that the covariance function is independent of t, thus we can define the autocovariance function:

$$\gamma_X(h) := \gamma(0,h) = \gamma(t,t+h)$$

and the autocorrelation function

$$\rho_X(h) := \frac{\gamma_X(h)}{\gamma_X(0)}$$

Other properties of the covariance function:

$$\gamma(0) \geq 0$$

$$\gamma(0) \geq |\gamma(h)| \text{ for all } h$$

$$\gamma(h) = \gamma(-h) \text{ for all } h$$

We can typically decompose a time series process X_t into

$$X_t = m_t + s_t + Y_t$$

where m_t is the long term trend component, s_t is the seasonal component and Y_t is a stationary random noise term. Given a realization x_t and estimates $\hat{m_t}$ and $\hat{s_t}$ we can find the estimated noise sequence by

$$\hat{Y_t} = x_t - \hat{m_t} - \hat{s_t}$$

2.2 AR & MA models

A first order autoregressive model (AR(1) model) $\{X_t\}$ is defined as

$$X_t = \phi X_{t-1} + Z_t,\ t = 0, \pm 1, \pm 2, ...$$

where $Z_t \sim WN(0, \sigma^2)$, $|\phi| < 1$ and Z_t is uncorrelated with X_s for all

$s < t$. The AR(1) model can also be expressed as

$$\begin{aligned} X_t &= \phi X_{t-1} + Z_t \\ &= \phi^2 X_{t-2} + \phi Z_{t-1} + Z_t \\ &= \phi^3 X_{t-3} + \phi^2 Z_{t-2} + \phi Z_{t-1} + Z_t \\ &= ... \\ &= \sum_{j=1}^{\infty} \phi^j Z_{t-j} \end{aligned}$$

which has the form of a MA(∞) model. An AR(p) model is of the form

$$X_t = \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + Z_t = \sum_{j=1}^{p} \phi_j X_{t-j} + Z_t$$

The autocorrelation function of an AR(1) function is

$$\rho(h) = \phi^h, \quad h \geq 0$$

A stationary solution of the AR(p) model exist if and only if

$$\phi(z) = 1 - \phi_1 z - ... - \phi_p z^p \neq 0 \quad \forall\, |z| = 1$$

Furthermore, the process is causal if

$$\phi(z) = 1 - \phi_1 z - ... - \phi_p z^p \neq 0 \quad \forall\, |z| \leq 1$$

Causality implies that the process $\{X_t\}$ is independent of future values of Z_s, where $s > t$. $\{X_t\}$ is weakly stationary if $\mu_X(t)$ is independent of t and if the covariace function $\gamma_X(t+h, t)$ is independent of t for all h.

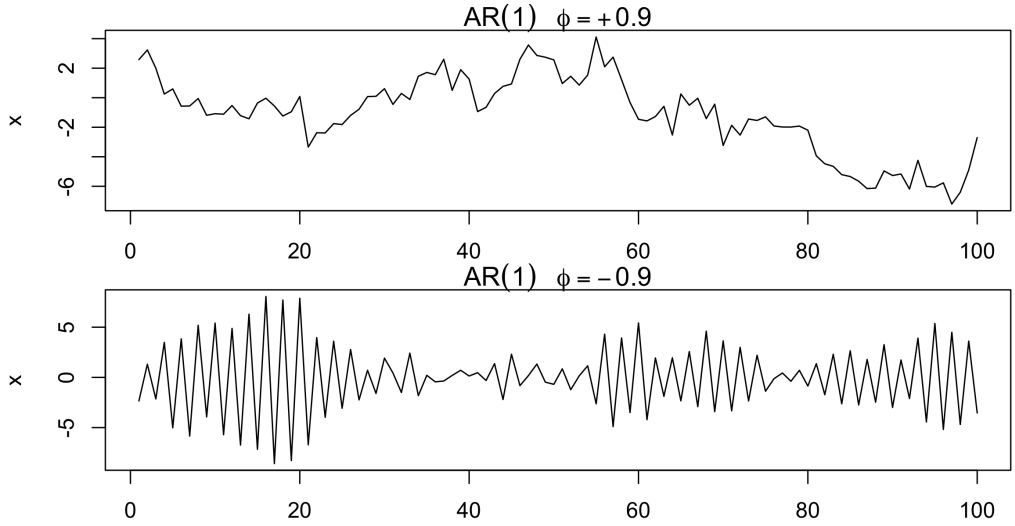


Figure 1: Simulated AR(1) model with $\phi = 0.9$ (top) and $\phi = -0.9$ (bottom)

A first order moving average (MA) model X_t is given by

$$X_t = \theta Z_{t-1} + Z_t$$

The MA(1) model can also be expressed as

$$\begin{aligned} X_t &= \theta Z_{t-1} + Z_t \\ &= -\theta X_{t-1} + \theta^2 Z_{t-2} + Z_t \\ &= -\theta X_{t-1} - \theta^2 X_{t-2} + \theta^3 Z_{t-3} + Z_t \\ &= ... \\ &= -\sum_{j=1}^{\infty} \theta^j X_{t-j} + Z_t \end{aligned}$$

which is the form of an AR(∞) model. The MA(q) model X_t is given by

$$X_t = \sum_{j=1}^{q} \theta_j Z_{t-j} + Z_t$$

The MA(q) model is said to be q-correlated, which means $\gamma(h) = 0$ for $|h| > q$. In fact every q-correlated process is a MA(q) process. The MA(1) model is said to be invertible if $|\theta| < 1$. The MA(q) model is invertible if

$$\theta(z) = 1 + \theta_1 z + ... + \theta_q z^q \neq 0 \text{ for all } |z| \leq 1$$

Invertibility allows us to express Z_t in terms of only current and previous values of X_s, $s \leq t$.

A first order autoregressive moving average (ARMA) model X_t is a combination of a an AR(1) and a MA(1) process, defined by

$$X_t = \phi X_{t-1} + \theta Z_{t-1} + Z_t$$

The ARMA(p,q) model is similarly

$$X_t = \sum_{j=1}^{p} \phi_j X_{t-j} + \sum_{j=1}^{q} \theta_j Z_{t-j} + Z_t$$

again $Z_t \sim WN(0, \sigma^2)$, $\phi_p \neq 0, \theta_q \neq 0$ and the process $\{X_t\}$ needs to be stationary. A unique and stationary solution $\{X_t\}$ exists if and only if

$$\phi(z) = 1 - \phi_1 z - ... - \phi_p z^p \neq 0 \quad \forall\, |z| = 1$$

The causality and invertibility conditions are also similar to the AR(p) and MA(q) models. A useful property of ARMA models is that for any positive integer k there exists an ARMA(p,q) process $\{X_t\}$ such that $\gamma_X(h) = \gamma(h)$ for $h = 0, 1, ..., k$.

The basic ARMA models are unfortunately not great for financial time series modelling as the conditional variance would be independent of time. Therefore, they can't capture volatility clustering. We now begin to introduce models that can.

2.3 (G)ARCH model

Financial time series are often represented as prices P_t of a stock, index, currency, commodity or interest rate. Sometimes we are most interested in the percentage return, given by the log return $Z_t = log\frac{P_t}{P_{t-1}} = log(1+r) = R$, where r is the continuous rate of return and R is the continuously compounded total return from $t-1$ to t. We introduce the conditional variance term h_t of Z_t, that will vary with time and allow us to better capture volatility clusters.

2.3.1 ARCH model

We now introduce the ARCH (Auto-regressive conditional heteroscedasticity) model (Engle 1982). The model is appropriate if the conditional error variance follows an AR model. ARCH models and its many variations have found a vast number of applications in financial time series modelling. The $ARCH(p)$ process $\{Z_t\}$ is given by

$$Z_t = e_t\sqrt{h_t}, \quad e_t \sim N(0,1)$$

where h_t is a positive function defined by

$$h_t = w + \sum_{j=1}^{p} \alpha_j Z_{t-j}^2$$

$w > 0, \alpha_j \geq 0, j = 1, ..., p$. If $\alpha = 0$ we simply get Gaussian white noise. Here $\{Z_t\}$ is a zero mean process, as is typical in financial return series. We can include a mean for Z_t by $x_t\xi$, where x_t is a vector of exogenous variables and ξ

is a vector of regression parameters. We can the let $\Psi_t = \{z_t, x_t, z_{t-1}, x_{t-1}, ...\}$ denote the information available at time t. We can then state the model more formally by

$$
\begin{aligned}
Z_t|\Psi_{t-1} &\sim N(x_t\xi, h_t) \\
h_t &= w + \sum_{j=1}^{p} \alpha_j Z_{t-j}^2 \\
e_t &= z_t - x_t\xi
\end{aligned}
$$

From here on we will only consider the zero mean model. In this model a large value of z_t of either sign will increase h_{t+1} so that large values of z_{t+1} of either sign tend to follow. Similarly small values of z_{t+1} tend to follow small values of z_t of either sign. The order of the lag p determines the amount of time volatility shocks persist in the conditional variance. A larger value of p will tend to increase the duration of volatility clusters. This model can be estimated by Maximum likelihood or ordinary least squares. The log-likelihood function is given by

$$l(\theta) = \frac{1}{2n} \sum_{t=1}^{n} -log(h_t) - \frac{e_t^2}{h_t}$$

We then maximize the log-likelihood function with respect to the model parameters $\theta = (w, \alpha_1, ..., \alpha_p)$. The Lagrange Multiplier test can be used to test for significance. The null-hypothesis is that all $\alpha_i = 0$. Even though the conditional error is normal, the unconditional error is not and has thicker tails. For the ARCH(1) model we can easily calculate

$$E(Z_t^2|Z_{t-1}) = (w + \alpha Z_{t-1}^2)E(e_t^2|Z_{t-1}) = w + \alpha Z_{t-1}^2$$

We see that the process $\{Z_t\}$ is not IID, and thus it is not normally distributed.

2.3.2 GARCH model

The generalized ARCH model (Bollerslev 1986) is appropriate when the error variance follows an ARMA model. The GARCH(p,q) model is then

$$Z_t = e_t\sqrt{h_t}, \quad e_t \sim IID(0,1)$$

where h_t is a positive function defined by

$$h_t = w + \sum_{j=1}^{p} \alpha_j Z_{t-j}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}$$

$w > 0, \alpha_j, \beta_j \geq 0, j = 1, 2, ...$

We can use $e_t \sim N(0,1)$, or we can use other $IID(0,1)$ distributions such as the standardized t-distribution

$$\sqrt{\frac{\nu}{\nu-2}} e_t \sim t_\nu, \ \nu > 2$$

Typically we only consider the GARCH(1,1) model where the conditional variance is

$$h_t = w + \alpha Z_{t-1}^2 + \beta h_{t-1}$$

where $w + \alpha + \beta = 1$ to ensure that the long run unconditional variance is equal to 1. This model incorporates previous values of the conditional variance in the form of a moving average. Similar to how a MA process can be viewed as an infinite order AR process, the GARCH process is an infinite order ARCH process. Thus, the GARCH process can sparsely represent a

high order ARCH process.

Bollerslev (1986) proved that the unconditional variance of the GARCH(p,q) is given by

$$Var(Z_t) = E(Z_t^2) = \frac{w}{1 - \sum_{j=1}^{p} \alpha_j + \sum_{j=1}^{q} \beta_j}$$

where $\sum_{j=1}^{p} \alpha_j + \sum_{j=1}^{q} \beta_j < 1$ is a necessary and sufficient condition for the the existence of the variance. Since Z_t is conditionally normal, $E(Z_t^m) = 0$ for all odd integers m. It follows that the skewness is zero, and that the unconditional distribution is symmetric. The kurtosis for a GARCH(p,q) is not easily available, but Engle (1982) and Bollerslev (1986) stated it for the ARCH(1) and GARCH(1,1) respectively. If $3\alpha^2 + 2\alpha\beta + \beta^2 < 1$ we have for the GARCH(1,1)

$$E(Z_t^4) = \frac{3w^2(1 + \alpha + \beta)}{(1 - \alpha - \beta)(1 - 3\alpha^2 - 2\alpha\beta - \beta^2)}$$

and

$$\begin{aligned} \frac{E(Z_t^4)}{E(Z_t^2)^2} &= \frac{3w^2(1 + \alpha + \beta)(1 - \alpha - \beta)^2}{w^2(1 - \alpha - \beta)(1 - 3\alpha^2 - 2\alpha\beta - \beta^2)} \\ &= 3\frac{(1 - \alpha^2 - \alpha\beta - \beta^2)}{(1 - 3\alpha^2 - 2\alpha\beta - \beta^2)} \end{aligned}$$

which is clearly greater than 3, since $\alpha, \beta \geq 0$. Hence the GARCH process is leptokurtic. If $\beta = 0$ in the above equation we get the kurtosis for the ARCH(1), which is also clearly leptokurtic, as long as $\alpha > 0$. For $k \geq 1$ the autocovariance of a GARCH process is given by

$$\begin{aligned} Cov(Z_t, Z_{t-k}) &= E[E(Z_t Z_{t-k}|\Psi_{t-1})] \\ &= E[Z_{t-k}E(Z_t|\Psi_{t-1})] = 0 \end{aligned}$$

so the GARCH process is serially uncorrelated. The process is therefore also weakly stationary if the variance exists. Since the serial correlation is zero past returns can't improve prediction of future returns, so there is no violation to the efficient markets hypothesis.

When we introduced the GARCH we did not require the conditional error to be normal. In practice the kurtosis of the conditional error often exceeds three, so that an assumption of normal errors is not appropriate. Bollerslev (1987) used a GARCH(1,1) with t-distributed conditional errors applied to U.S. dollar versus British pounds and Deutschemark and to the S&P 500. The sample kurtosis of the residuals was around 4 for most of the datasets, far in excess of the normal kurtosis of three, but very close to the implied kurtosis from the fitted t-distribution. Although the t-distribution might fix overall conditional kurtosis it still assumes constant conditional kurtosis, which is not necessarily the case in practice. It also assumes no skewness of the conditional errors, but a skewed t-distribution can be used if there is a problem with skewed residuals. When the conditional error is non-normal we can still use the same log-likelihood function as in the ARCH model estimation to get estimates of the model parameters. These estimates are called the quasi maximum likelihood estimates (QMLE). The asymptotic distribution of the QML estimate $\hat{\theta}$ is then, under certain regularity conditions, given by (Bollerslev and Wooldridge 1992)

$$\sqrt{n}(\hat{\theta} - \theta_0) \sim N(0, A^{-1}BA^{-1})$$

where θ_0 is the true parameter values. Consisten estimators of A and B are

given by

$$\hat{A} = -\frac{\partial^2 l(\hat{\theta})}{\partial\theta\ \partial\theta'} \text{ and } \hat{B} = \frac{\partial l(\hat{\theta})}{\partial\theta}\frac{\partial l(\hat{\theta})'}{\partial\theta}$$

The matrix $\hat{A}^{-1}\hat{B}\hat{A}^{-1}$ is a consistent estimate of the asymptotic variance matrix of $\sqrt{n}(\hat{\theta} - \theta_0)$. If the residual distribution is normal, then $A = B$, and the covariance matrtix estimator can be found using either $\hat{A}^{-1}$ or $\hat{B}^{-1}$.

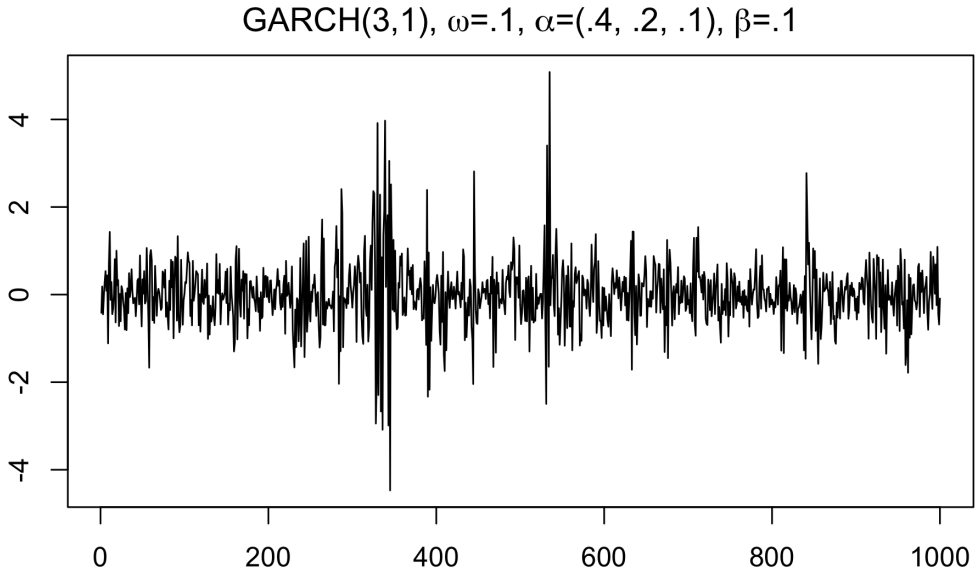


Figure 2: Simulated GARCH(3,1) model

We have seen that the GARCH model has a zero mean, zero autocorrelation, heavy tails and can capture volatility clustering and persistence. It is therefore a very useful model for financial return series modelling. We see that plot of the simulated return series looks a lot like the observed return series in Section 5. Large fluctuations in prices tends be followed by more volatility. Volatility can also stay suppressed for long times before "exploding". The GARCH model assumes linear relationship between return and volatility. As discussed earlier negative news generally impacts volatility of financial assets

more than positive news. The (G)ARCH model does however give equal weight to positive and negative price fluctuations and is unable to capture the leverage effect. We will make another extension to capture this asymmetry. We will also see that introducing a power transformation of the absolute return lets us better capture the long-term memory of stock returns.

2.4 APARCH models

The APARCH(1,1) model (asymmetric power ARCH), Ding *et al* (1993), is

$$Z_t = e_t\sqrt{h_t}, \quad e_t \sim IID(0,1)$$
$$h_t^{\delta/2} = w + \alpha(|Z_{t-1}| - \gamma Z_{t-1})^\delta + \beta h_{t-1}^{\delta/2}$$

where $w, \alpha, \beta > 0, \delta \geq 0, \ -1 < \gamma < 1$. The APARCH(p,q) model is similarly

$$h_t^{\delta/2} = w + \sum_{i=1}^{p} \alpha_i(|Z_{t-1}| - \gamma_i Z_{t-1})^\delta + \sum_{j=1}^{q} \beta_j h_{t-1}^{\delta/2}$$

where $w > 0, \ \delta \geq 0, \ \alpha_i \geq 0$ and $-1 < \gamma_i < 1$, for $i = 1, ..., p$ and $\beta_j \geq 0$ for $j = 1, ..., q$. The asymmetry coefficient γ controls the different response in volatility depending on positive and negative returns. This effect is well documented is finance, and is called the leverage effect. We see that when $\gamma > 0$, which it will be for most financial return series, a negative value of Z_{t-1} will give a larger value of h_t than a positive value of Z_{t-1} of the same magnitude. δ is the power coefficient, allowing different powers of transformation. The stationary condition for the APAPRCH(1,1) model is $\alpha(1 + \gamma^2) + \beta < 1$. Using (quasi) Maximum Likelihood we can estimate the parameter vector $\hat{\boldsymbol{\theta}} = (\hat{\alpha}, \hat{\beta}, \hat{\delta}, \hat{\gamma}, \hat{w})$. Then for current values of Z_{t-1} and $\hat{h}_{t-1}$ we can make a forecast of $\hat{h}_t$ by

$$\hat{h}_t^{\delta/2} = \hat{w} + \hat{\alpha}(|Z_{t-1}| - \hat{\gamma} Z_{t-1})^{\hat{\delta}} + \hat{\beta}\hat{h}_{t-1}^{\hat{\delta}/2}$$

We notice that $\delta = 2, \gamma = 0$ gives the GARCH model. Other popular models includes the TS-GARCH, Taylor (1986), Schwert (1989), with $\delta = 1, \gamma = 0$,

$$h_t^{1/2} = w + \alpha|Z_{t-1}| + \beta h_{t-1}^{1/2}$$

The T-GARCH, Zakoian(1994), with $\delta = 1$

$$h_t^{1/2} = w + \alpha(|Z_{t-1}| - \gamma Z_{t-1}) + \beta h_{t-1}^{1/2}$$

And the GJR-GARCH, Glosten, Jagannathan and Runkle (1993), with $\delta = 2$

$$h_t = w + \alpha(|Z_{t-1}| - \gamma Z_{t-1})^2 + \beta h_{t-1}$$

We observe that the GARCH and TS-GARCH models does not incorporate the asymmetry effect, while the T-GARCH and GJR-GARCH models do. A nice property of these models is that many of them are nested, and can thus be compared by a likelihood ratio test. Ding *et al* rejected both the GARCH and the TS-GARCH models in favor of the APARCH model when applied to the S&P 500. The estimated value of δ and γ in their experiment was 1.43 and 0.373 respectively, both very significantly. The estimation of the models are based on Maximum Likelihood when the conditional errors follow the normal distribution and Quasi-Maximum Likelihood (QML) for other distributions. A skewed Student's t-distribution is proposed by Fernández and Steel (1998) to accommodate asymmetry and kurtosis in the error term of regression problems. When the underlying distribution exhibit increasing

skewness the QML will be increasingly ineffective (Engle and González-Rivera, 1991). To potentially improve model fitting in instances of skewed underlying distributions we introduce Support Vector Machine based regression.

3 Support Vector Regression

Support vector machine's (SVM) are supervised learning models used mostly for classification purposes. Support vector machines were introduced by Vapnik *et al* (1992). They are based on earlier work Vapnik, cited in the introduction. SVMs are one of the most powerful supervised learning classifiers. A large range of applications make use of SVMs. Support vectors can also be used for unsupervised clustering, see *Support vector clustering*, Vapnik et al (2001). The support vector algorithms are entirely data driven, and need no assumptions of the properties of the underlying distribution. The SVM can be computationally sparse and achieve good accuracy even with limited sample sizes. This is a result of the SVM's kernel based methodology, which allows for non-linear relationships. The ϵ-insensitive loss function, introduced by Vapnik (1995) allowed for extension of the SVM framework to regression problems. The support vector machines are non-parametric and are not useful for statistical inference. Their purpose is generally forecasting, and empirical performance indicate strong performance. This chapter presents the linear maximal margin classifier, extensions to non-linear classifiers, ie. SVMs, and finally Support vector regression.

3.1 Maximal Margin Classifier

The maximal margin classifier was introduced by Vapnik and Lerner (1963) as a way to classify data as one of two separable classes. We need the data to be linearly separable to be able to use the maximal margin classifier. Suppose we have a $n \times p$ data matrix $\mathbf{X}$ with n observations of p variables. We want to separate each observation into one of two classes. To do this we create a $p-1$ dimensional hyperplane so that each observation is classified according to which side of the hyperplane it resides. Given two classes for classification, we represent them as $y_i \in \{-1, 1\}$ for $i = 1, ..., n$. Then given the coefficient vector $\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_p)$ a separating hyperplane have the properties

$$\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i > 0 \text{ if } y_i = 1$$

and

$$\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i < 0 \text{ if } y_i = -1$$

or, equivalently

$$y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) > 0$$

Then the formula for the seperating hyperplane will be

$$f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x} = 0$$

The hyperplane in a 2-dimensional space will be a line and and in a 3-dimensional space be a plane. If the data can be completely separated, there will exist infinitely many separating hyperplanes. To select the best hyperplane we impose the constraint that the hyperplane needs to be at a maximal distance from the classes. This leads us to the maximal margin hyperplane, the hyperplane with the largest minimum distance to all/both classes of observations. We measure the perpendicular distance between observation i and the hyperplane by

$$m_i = |\frac{f(x_i)}{||\beta||}|$$

We then want to find the the observations closest to the separating hyperplane

$$M = \underset{i=1,\ldots,n}{min}\ m_i$$

The points closest the hyperplane on each side of the hyperplane are the support vectors. The hyperplane is supported by these observations, and

in fact, the hyperplane only depends on the support vectors. If one of the support vectors change, so will the hyperplane. Since predictions only depend on the support vectors for classification, this is computationally easy. A parallel line to the separating hyperplane runs through the support vectors on each side. These lines are called the margin lines. Changes in the observations outside of the margin lines does not change the separating hyperplane, as long as they do not cross the margin lines. If a point is moved inside the margin lines, that point will becom a support vector and the hyperplane will change accordingly.

The maximum margin hyperplane is the solution to the following optimization problem

$$\begin{aligned} &\underset{\beta_0,\boldsymbol{\beta}}{\text{maximize}}\ M \\ &\text{subject to}\ \begin{cases} \sum_{j=1}^{p} \beta_j^2 = 1 \\ y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) \geq M\ \ i = 1, ..., n \end{cases} \end{aligned}$$

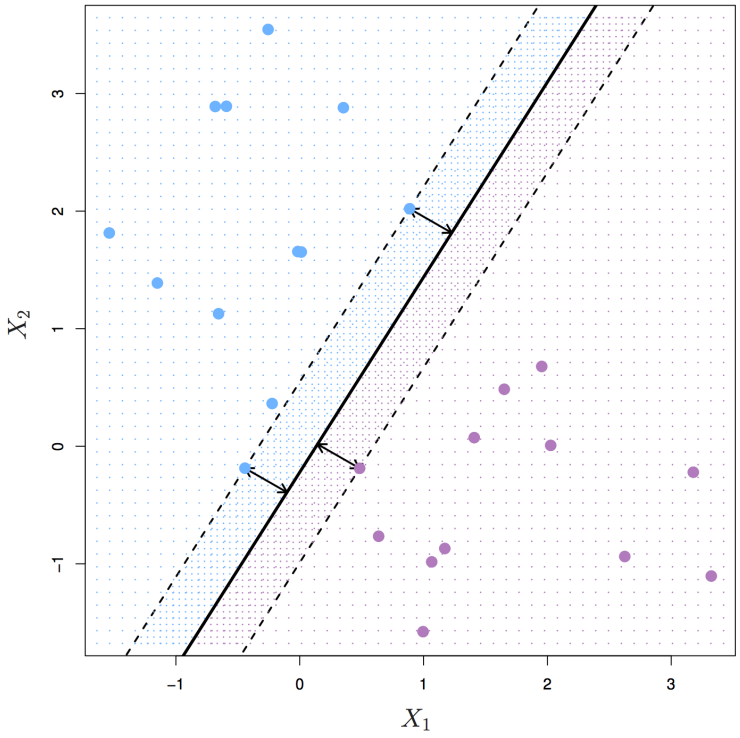


Figure 3: Maximal margin classifier with two classes. From James et al. (2013)

The second constraint makes sure that every observation is in the correct side of the hyperplane. With the first constraint we get that the perpendicular distance from the ith observation to the hyperplane is given by $y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i)$. Combined these constraints makes sure that every observation is on the correct side of the hyperplane and at least a distance M from the hyperplane, as desired. Dividing both the constraint and M by $||\boldsymbol{\beta}||$ we get rid of the first constraint. We then rescale the parameters $\beta_0, \boldsymbol{\beta}$ such that $M = 1$. The optimization problem can then be restated as

$$\begin{aligned}&\underset{\beta_0,\boldsymbol{\beta}}{\text{minimize}}\ \frac{1}{2}||\boldsymbol{\beta}||^2\\&\text{subject to } y_i(\beta_0+\boldsymbol{\beta}^\intercal\boldsymbol{x}_i)-1\geq 0\ i=1,...,n\end{aligned}$$

This is a convex quadratic minimization problem with inequality constraints. The optimization problem has the form

$$\begin{aligned}&\underset{\boldsymbol{x}}{\text{minimize}}\quad f(\boldsymbol{x})\\&\text{subject to } g(\boldsymbol{x})>0\end{aligned}$$

This problem can be solved by Lagrange multipliers. The method of Lagrange multipliers, of the 18th century mathematician Joseph-Louis Lagrange, states that in order to find the maximum or minimum of a function $f(x)$ subjected to the equality constraint $g(x)=0$, form the Lagrange function $\mathcal{L}(\alpha,x)=f(x)-\alpha g(x)$ and solve $\nabla\mathcal{L}(\alpha,x)=0$. The Lagrangian dual problem is

$$\begin{aligned}&\underset{\alpha}{\text{maximize}}\ \underset{\beta_0,\boldsymbol{\beta}}{\text{minimize}}\mathcal{L}(\alpha,\beta_0,\boldsymbol{\beta})=f(\boldsymbol{x})-\sum_{i=1}^{n}\alpha_i g(\boldsymbol{x})\\&\text{subject to } \alpha_i\geq 0\quad\forall i=1,...,n\end{aligned}$$

where the Lagrange function of our optimization problem is

$$\mathcal{L}(\alpha,\beta_0,\boldsymbol{\beta})=\frac{1}{2}||\boldsymbol{\beta}||^2-\sum_{i=1}^{n}\alpha_i[y_i(\beta_0+\boldsymbol{\beta}^\intercal\boldsymbol{x}_i)-1]$$

and the parameters to be determined, $\beta_0,\boldsymbol{\beta}$, are called the primal variables, while the $\alpha_i,\ i=1,...,n$, are the Lagrange multipliers. The Lagrange

multipliers restrict the space of values feasible for a solution, given the constraints. Since we have an inequality constraint, the Karush-Kuhn-Tucker (1939, 1951) (KKT) conditions must be satisfied to generalize the Lagrange multipliers. The conditions are

Primal constraint

$$y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) - 1 \geq 0 \quad \forall i = 1, ..., n$$

Dual constraint

$$\alpha_i \geq 0 \quad \forall i = 1, ..., n$$

Complementary slackness

$$\alpha_i[y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) - 1] = 0 \quad \forall i = 1, ..., n$$

Gradient of Lagrangian

$$\frac{\partial \mathcal{L}(\alpha, \beta_\prime, \boldsymbol{\beta})}{\partial \beta_0} = \sum_{i=1}^{n} \alpha_i y_i = 0$$
$$\frac{\partial \mathcal{L}(\alpha, \beta_\prime, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0$$

The complementary slackness is the relationship between the primal and the dual constrains, so that we get equalities. We see that the Lagrange multiplier $\alpha_i = 0$ for all $y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) - 1 > 0$. This means only points on the margin influence the optimization. To solve the inner minimization problem of the dual problem we use the gradient of the Lagrangian. Substituting the gradient of the Lagrangian into the Lagrange function we get the Wolfe dual problem

$$\underset{\alpha}{\text{maximize}}\ \mathcal{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\intercal \boldsymbol{x}_j$$

$$\text{subject to } \begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \alpha_i \geq 0 \quad i = 1, ..., n \end{cases}$$

The second constraint removes β_0 from the optimization problem. Thus, the Wolfe dual problem only depends on the Lagrange multipliers α. Also the training data is only used to compute the inner products between observations, which will come in handy later on with SVMs. Solving the Wolfe dual problem we get the solution for the Lagrange multipliers α. We can compute $\boldsymbol{\beta}$ from the gradient of the Lagrangian

$$\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$$

Then β_0 can be calculated as the average difference between observations and function values for the set of support vectors S

$$\begin{aligned} \beta_0 &= \frac{1}{S} \sum_{k \in S} (y_k - \boldsymbol{\beta}^\intercal \boldsymbol{x}_k) \\ &= \frac{1}{S} \sum_{k \in S} (y_k - \sum_{j=1}^{n} \alpha_j y_j \langle \boldsymbol{x}_j, \boldsymbol{x}_k \rangle) \end{aligned}$$

The classification of new data will then be the sign of the function

$$f(\boldsymbol{x}) = \beta_0 + \sum_{k \in S} \alpha_k y_k \langle \boldsymbol{x}_k, \boldsymbol{x} \rangle$$

3.2 Support Vector Classifier

If the classes are not perfectly separable, the maximal margin optimization has no solution. We solve this problem by a soft margin classifier. The soft margin hyperplane classifier was developed by Vapnik and Cortes in 1993 (Cortes and Vapnik, 1995). This classifier allows some of the observations to be on the wrong side of the margin, and even in the wrong side of the separating hyperplane. This will reduce sensitivity to outliers and make the sacrifice of misclassifying some observations in return for better classification of most of the observations. This method is called the support vector classifier, or soft margin classifier. We introduce the slack variables $\xi_i = \xi_1, ..., \xi_n,\ \xi_i \geq 0,\ \forall\, i$. An $\xi_i = 0$ indicates an observation on the correct side of the margin lines, while $\xi_i > 0$ indicates an observation on the wrong side of the margin. $\xi_i = 1$ an observation on the hyperplane, and $\xi_i > 1$ indicates that observation i is on the wrong side of the hyperplane. The classification hyperplane is the solution to the optimization problem

$$\underset{\beta_0, \boldsymbol{\beta}, \boldsymbol{\xi}}{\text{maximize}}\ M$$

$$\text{subject to} \begin{cases} \sum_{j=1}^{p} \beta_j^2 = 1 \\ y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) \geq M(1 - \xi_i)\ i = 1, ..., n \\ \xi_i \geq 0 \\ \sum_{i=1}^{n} \xi_i \leq C \end{cases}$$

C here is a non-negative tuning hyperparameter, and gives an upper bound to the sum of the slack variables. A larger value of C allows for more and larger violations of the margin, which widens the margin. This will increase the bias of hyperplane, but reduce the variance of the solution. A small value of C will increase the variance of the solution but it will be less biased. Setting $C = 0$ results in the maximum margin hyperplane. The optimal

value of C can be chosen by cross-validation. As with the maximal margin classifier, only a subset of the observations will affect the hyperplane. Those observations are the ones on or on the wrong side of the margin, the support vectors. Figure 4 shows the Support Vector classifier with different values of the hyperparameter C. We can see that the two classes are not seperable, and how some of the observations cross the margin lines and the hyperplane. The top left panel shows the Support Vector classifier with a large value of C. We see that the margins are wide, and we have many observations on the inside of the margins. These observations are the support vectors. The bottom right panel shows a small value of C, and we get tight margin lines and fe support vector. We reformulate the optimization problem in a similar fashion as in the last section, by rescaling the parameters, and obtain the new optimization problem

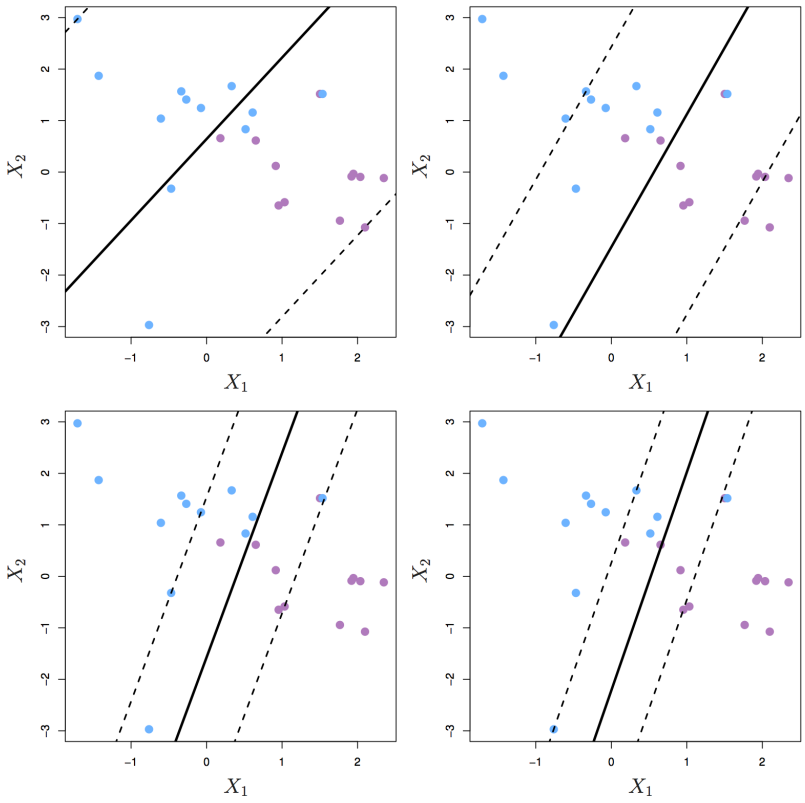


Figure 4: Support Vector classifier with different values of the hyperparameter C. From James et al. (2013)

$$\underset{\beta_0,\boldsymbol{\beta},\boldsymbol{\xi}}{\text{minimize}} \ \frac{1}{2}||\boldsymbol{\beta}||^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \begin{cases} y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) \geq 1 - \xi_i & i = 1, ..., n \\ \xi_i \geq 0 & i = 1, ..., n \end{cases}$$

The Lagrange function for this problem is

$$\mathcal{L}(\alpha, \eta, \beta_0, \boldsymbol{\beta}, \xi) = \frac{1}{2}||\boldsymbol{\beta}||^2 - \sum_{i=1}^{n} \alpha_i[y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) - (1 - \xi_i)] + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \eta_i \xi_i$$

where α, η are the Lagrange multipliers. We obtain the Lagrange dual problem

$$\begin{aligned} &\underset{\alpha,\eta}{\text{maximize}}\ \underset{\beta_0,\boldsymbol{\beta}}{\text{minimize}} \mathcal{L}(\alpha, \eta, \beta_0, \boldsymbol{\beta}, \xi) \\ &\text{subject to } \alpha_i, \eta_i \geq 0 \quad \forall i = 1, ..., n \end{aligned}$$

The corresponding KKT conditions are

$$\begin{aligned} \alpha_i, \eta_i, \xi_i \geq 0 \quad &\forall\, i = 1, ..., n \\ \eta_i \xi_i = 0 \quad &\forall\, i = 1, ..., n \\ y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) - (1 - \xi) \geq 0 \quad &\forall\, i = 1, ..., n \\ \alpha_i[y_i(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) - (1 - \xi)] = 0 \quad &\forall\, i = 1, ..., n \end{aligned}$$

and the gradients of the Lagrangian are

$$\begin{aligned} \frac{\partial}{\partial \beta_0} \mathcal{L}(\alpha, \eta, \beta_0, \boldsymbol{\beta}, \xi) &= \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \frac{\partial}{\partial \boldsymbol{\beta}} \mathcal{L}(\alpha, \eta, \beta_0, \boldsymbol{\beta}, \xi) &= \boldsymbol{\beta} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0 \\ \frac{\partial}{\partial \xi_i} \mathcal{L}(\alpha, \eta, \beta_0, \boldsymbol{\beta}, \xi) &= C - \alpha_i - \eta_i = 0 \end{aligned}$$

The inner minimization problem is agian solved by substituting in the gradients of the Lagrangian with respect to the primal variables. And thus the Wolfe dual problem is

$$\begin{aligned} &\underset{\alpha}{\text{maximize}} \quad \mathcal{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\intercal \boldsymbol{x}_j \\ &\text{subject to} \begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C \quad i = 1, ..., n \end{cases} \end{aligned}$$

where the condition $\alpha_i \leq C$ is implied by the gradient of the Lagrangian with respect to ξ, $\alpha_i = C - \eta_i$ since $\eta_i \geq 0$. This constraint is called a box constraint. We notice that we are able to get rid of η in the optimization, and the only additional task compared to the maximal margin classifier is to determine the value of C. The points where $\alpha_i = 0$ have $\eta_i = C$ so that $\xi_i = 0$. Thus, these points are on the correct side of the margin and don't influence the model. If $0 < \alpha_i < C$ then $0 < \eta_i < C$ so that again $\xi_i = 0$, hence the point lies on the margin. If $\alpha_i = C$ then $\xi_i > 0$, meaning the observation is on the wrong side of the margin and maybe even on the wrong side of the of the hyperplane. The point will be on the correct side of the hyperplane if $0 < \xi_i \leq 1$ and on the wrong side if $\xi_i > 1$. In sum the KKT conditions assures that only support vectors affect the solution so that the solution is sparse. Another great property is the ability to control the bias-variance tradeoff through the hyperparameter C. For these reasons the support vector classifier is a great method for linear classifications. Again, the prediction of new data points will be the sign of

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle$$

3.3 Support Vector Machines

If the boundary between classes is non-linear the support vector classifier will perform poorly. Using kernel functions we can enlarge the feature space to enable non-linear boundaries. The method is known as support vector machines. The idea is to apply the mapping function ϕ to the data, $(\phi(\boldsymbol{x}_1), ..., \phi(\boldsymbol{x}_n)))$, and then use the support vector classifier. We use ϕ map the p-dimensional training data from the space X into a higher dimensional feature space Y where we can divide the observations by a linear hyperplane, and map back the non-linear classification boundary to the original space. Instead of explicitly applying the transformations we use kernel functions to return the inner product in the feature space. This procedure is called the kernel trick (Schölkopf and Smola, 2002).

A kernel is a similarity function of data points. Kernel functions allows operations in the implicit higher-dimensional feature space Y without explicit computation of coordinates in that space. We only need to compute the inner products of the data in the higher dimension. This follows from Mercer's theorem (Mercer, 1909). The kernel function is a positive definite and symmetric function

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

where ϕ: $X \to Y$ is a mapping function and Y is a inner product space. In fact any function K that is positive definite and symmetric function is kernel function, with symmetry and positive definite defined as

$$K(x, y) = K(y, x)$$
$$\int \int g(x) K(x, y) g(y) dx dy \geq 0$$

respectively, for some function $g \in L^2$. The objective function of the dual problem will then be similar to the support vector classifier, only the inner product needs to be substituted by the kernel function:

$$\begin{aligned}\mathcal{L}(\alpha) &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)\end{aligned}$$

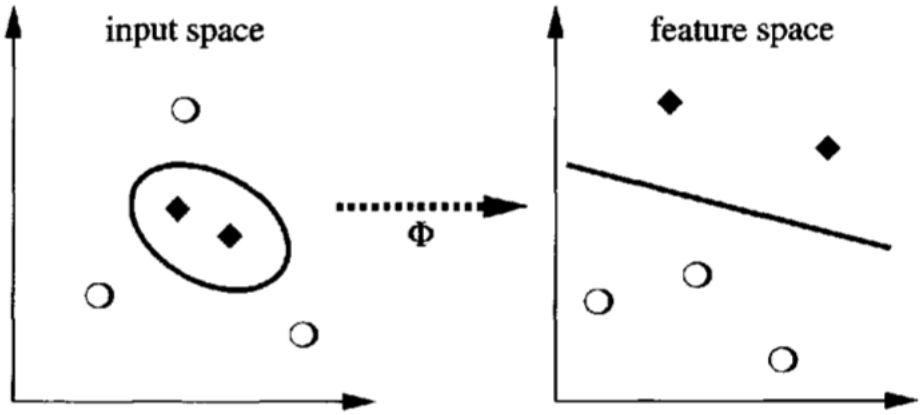


Figure 5: Support vector machines map the training data into a higher-dimensional feature space. From Schölkopf and Smola (2002)

Some popular choices of kernels include:

Linear kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}) = \boldsymbol{x}_i^\intercal \boldsymbol{x}$$

Polynomial kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}) = (1 + \boldsymbol{x}_i^\intercal \boldsymbol{x})^d$$

Radial kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}) = exp(-\gamma||\boldsymbol{x}_i - \boldsymbol{x}||^2)$$

Sigmoid kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}) = tanh(\nu + \gamma \boldsymbol{x}_i^\intercal \boldsymbol{x})$$

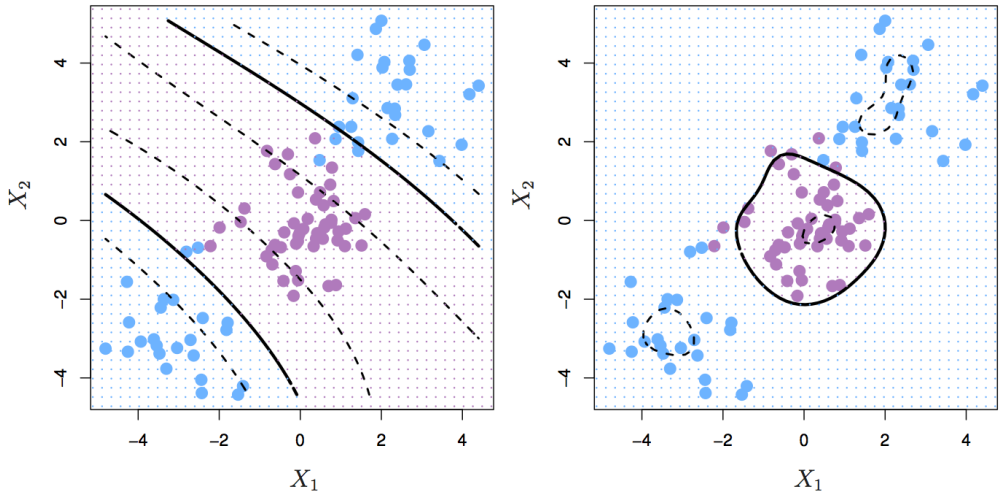


Figure 6: Support vector machines, with a polynomial kernel to the left and a radial kernel to the right. From James et al. (2013).

For the polynomial kernel, d is the degree of the polynomial. For the radial kernel, setting $\gamma = 1/2\sigma^2$ leads us to the (radial) Gaussian kernel. σ will function as a scale parameter for the Gaussian kernel, and determines the width of the kernel. For a test observation $\boldsymbol{x}$ the Gaussian kernel will give large weight to training observation $\boldsymbol{x}_i$ where the Euclidean distance $||\boldsymbol{x}_i - \boldsymbol{x}||$ is small. Training observations with larger Euclidean distance will get much smaller weighting, and thus contribute much less to classification of the test observation $\boldsymbol{x}$. The feature space of the Gaussian kernel is infinite-dimensional, so explicit calculations would be impossible. The Sigmoid kernel does not actually satisfy Mercer's condition but has nonetheless performed well in practice. If we let S be the collection of the support vector observation's

indices, then the function for the linear support vector classifier is

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i \in S} \alpha_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle = 0$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product, and the parameters $\alpha_i \neq 0 \; \forall \, i \in S$. In the general case with kernel $K(\boldsymbol{x}_i, \boldsymbol{x})$ we have the classifier function

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i \in S} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) = 0$$

and the predicted classification of data point $\boldsymbol{x}$ will be the sign of

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i \in S} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})$$

The SVM is a great method for non-linear classification. It is entirely data driven, it has a convex optimization giving a unique solution and is good for predictions. There is, however, a number of potential issues that must be resolved to use support vector machines. One must decide upon which kernel to use, and it is difficult to know beforehand which is most appropriate. Further the values of the hyperparameter C and ϵ has to be chosen. These values are important to balance the bias-variance tradeoff. A complex model will fit the training data well but will have large variance. It might be an overfit that performs poorly on test data. On the other end of the spectrum is very simple models. These have low variance but might be biased and fail to capture the basic structure of the data. The hyperparameter values are typically selected by cross-validation. The input data needs to be labeled to train the model. This can be a laborious or practically impossible task in reality. Support vector machines are entirely data driven, and thus does not incorporate base rate probabilities for classification. It is not straight forward to extend support vector machines to multiple classes, although extensions

exist. A major issue with support vector machines is the interpretability of model parameters. As the data often is transformed into multidimensional spaces to determine the relationships between input and output data, the support vector machine is considered a black box method.

3.4 Support Vector Regression

The SVM can be generalized to be applicable in regression problems. Support vector regression (SVR) was proposed by Vapnik et al. in 1995. As with SVMs, SVR use kernels and depends only an a subset of the data as support vectors. We now have input/output data set $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)$, where $\boldsymbol{x} = \boldsymbol{x}_1, ..., \boldsymbol{x}_n \in \mathbb{R}^p$ and $\boldsymbol{y} = y_1, ..., y_n \in \mathbb{R}$. We introduce the ϵ-insensitive region around the regression function, called the ϵ-tube. Our goal is to find the smoothest function that has at most ϵ deviation from the outputs $\boldsymbol{y}$. Similar to the soft-margin classifier we may allow some observations to fall outside of the ϵ-tube. Predictions that are farther than ϵ from their observed values will be penalized in the optimization. A larger value of ϵ will make the the ϵ-tube wider and increase the tolerance for error. Fewer observations will then reside outside the insensitive region. This gives lower variance, but higher bias of the solution. Conversely, a smaller ϵ, ie. a tighter ϵ-tube, will decrease the tolerance for errors, increase the number of support vectors, increase the variance and decrease the bias of the solution. To allow for some violations of the ϵ-tube we also have a tunable regularization cost hyperparameter C. Drucker et al. (1996) argues that SVR has greatest use when the dimensionality of the input space and the order of the approximation creates a dimensionality of a feature space representation that is large. We want to estimate a function of the form

$$f(x) = \beta_0 + \boldsymbol{\beta}^\intercal x$$

The smoothness of the function can be achieved by small regression coefficients $\boldsymbol{\beta}$. Thus we want to minimize $||\boldsymbol{\beta}||^2$. At the same time we want to penalize observations outside of the ϵ-tube. We do this by introducing the loss function $L(y_i, f(\boldsymbol{x}_i))$. The loss function should be convex to ensure an efficient solution to the optimization problem. The loss functions presented here are also symmetric, but they need not to be. For example, an asset manager would be far more concerned about large losses than large gains in their portfolio returns, and therefore penalize negative violations more than positive violations of the ϵ-tube. Some loss functions are

Linear

$$L(y_i, f(\boldsymbol{x}_i)) = \begin{cases} |y_i - f(\boldsymbol{x}_i)| - \epsilon & \text{if } |y_i - f(\boldsymbol{x}_i)| > \epsilon \\ 0 & \text{otherwise} \end{cases}$$

Quadratic

$$L(y_i, f(\boldsymbol{x}_i)) = \begin{cases} (|y_i - f(\boldsymbol{x}_i)| - \epsilon)^2 & \text{if } |y_i - f(\boldsymbol{x}_i)| > \epsilon \\ 0 \quad \text{otherwise} \end{cases}$$

Huber

$$L(y_i, f(\boldsymbol{x}_i)) = \begin{cases} c|y_i - f(\boldsymbol{x}_i)| - (c^2/2) & \text{if } |y_i - f(\boldsymbol{x}_i)| > c \\ \frac{1}{2}|y_i - f(\boldsymbol{x}_i)|^2 & \text{if } |y_i - f(\boldsymbol{x}_i)| \leq c \end{cases}$$

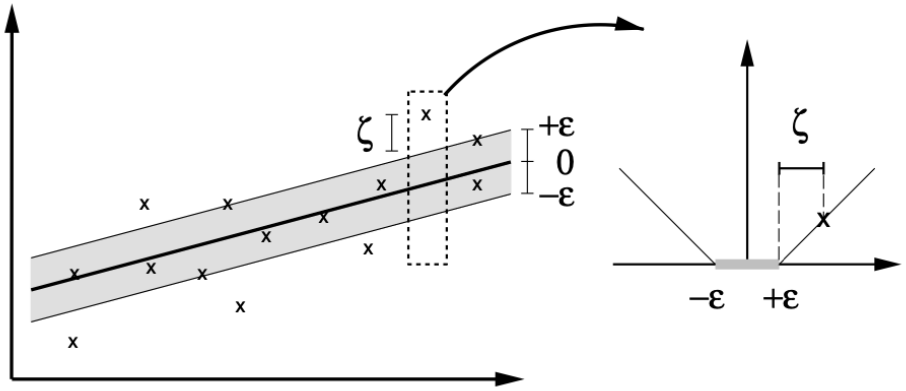


Figure 7: Example of support vector regression with linear loss function. From Schölkopf and Smola (2002)

Thus we want to minimize

$$\frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n} L(y_i, \boldsymbol{x}_i)$$

where the first term controls the smoothness of the function and the second term is a penalization term. In the following we will use the linear loss function. We introduce the slack variables ξ_i, ξ_i^* to denote the linear errors outside the ϵ-tube, to the downside and upside respectively. Then we state the optimization problem as

$$\underset{\boldsymbol{\beta},\boldsymbol{\xi},\boldsymbol{\xi}^*}{\text{minimize}}\ \frac{1}{2}||\boldsymbol{\beta}||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

$$\text{subject to}\ \begin{cases} y_i - \beta_0 - \boldsymbol{\beta}^\intercal \boldsymbol{x}_i \leq \epsilon + \xi_i \\ \beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \quad i = 1, ..., n \end{cases}$$

The hyperparameter C controls the tradeoff between the amount of empirical error to be tolerated and the flatness of the tube. The value of C has implications for the fit to training data, as well as to make reliable predictions. A larger value of C will increase the emphasis given to minimizing violations of the ϵ-tube, while a smaller value of C puts more emphasis on a flatter function. The values of the hyperparameters C and ϵ are typically chosen by cross-validation. As in the case of support vector machines, we can use kernels to implicitly map a non-linear functions into a higher dimension where a linear regression is applicable. The support vector regression only depends on the inner products of the support vectors in the higher dimension space. We construct a Lagrange function from the above objective function and constraints by introducing dual variables. This function has a saddle point with respect to the primal and dual variables at the solution. The resulting Lagrange function from the primal objective function can be expressed as:

$$\mathcal{L}(\alpha, \alpha^*, \eta, \eta^*; \beta_0, \boldsymbol{\beta}, \xi, \xi^*) = \frac{1}{2}||\boldsymbol{\beta}||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta_i^*\xi_i^*)$$
$$- \sum_{i=1}^{n}\alpha_i(\epsilon + \xi_i - y_i + \beta_0 + \boldsymbol{\beta}^\intercal\boldsymbol{x}_i) - \sum_{i=1}^{n}\alpha_i^*(\epsilon + \xi_i^* + y_i - \beta_0 - \boldsymbol{\beta}^\intercal\boldsymbol{x}_i)$$

where the Lagrange multipliers $\alpha, \alpha^*, \eta, \eta^* \geq 0$. By the saddle point conditions the partial derivatives with respect to the primal variables $\beta_0, \boldsymbol{\beta}, \xi_i, \xi_i^*$ must be zero

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0$$
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)x_i = 0$$
$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \eta_i - \alpha_i = 0$$
$$\frac{\partial \mathcal{L}}{\partial \xi_i^*} = C - \eta_i^* - \alpha_i^* = 0$$

and the partial derivatives with respect to the Lagrange multipliers $\alpha, \alpha^*, \eta, \eta^*$ gives the constraints

$$\frac{\partial \mathcal{L}}{\partial \eta_i} = \sum_{i=1}^{n} \xi_i \geq 0$$
$$\frac{\partial \mathcal{L}}{\partial \eta_i^*} = \sum_{i=1}^{n} \xi_i^* \geq 0$$
$$\frac{\partial \mathcal{L}}{\partial \alpha_i} = \epsilon + \xi_i - y_i + \beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i \geq 0$$
$$\frac{\partial \mathcal{L}}{\partial \alpha_i^*} = \epsilon + \xi_i^* + y_i - \beta_0 - \boldsymbol{\beta}^\intercal \boldsymbol{x}_i \geq 0$$

Substituting the partial derivatives with respect to the primal variables back into the Lagrange function the Wolfe dual optimization problem is given by

$$\underset{\alpha,\alpha^*}{\text{minimize}}\ \frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i, x_j\rangle - \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) + \epsilon\sum_{i=1}^{n}(\alpha_i - \alpha_i^*)$$
$$\text{subject to } \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \text{ and } \alpha_i, \alpha_i^* \in [0, C],\ i = 1, ..., n$$

Agian we are able to get rid of the η, η^* Lagrange multipliers and are left with α, α^*. The KKT conditions must be satisfied since the optimization is under inequality constraints. The conditions require that the product between dual variables and constraints disappear at the point of the solution. Thus we have

$$\alpha_i(\epsilon + \xi_i - y_i + \beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) = 0$$

$$\alpha_i^*(\epsilon + \xi_i^* + y_i - \beta_0 - \boldsymbol{\beta}^\intercal \boldsymbol{x}_i) = 0$$

$$(C - \alpha_i)\xi_i = 0$$

$$(C - \alpha_i^*)\xi_i^* = 0$$

For points where $|\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}_i - y_i| < \epsilon$ we need $\alpha_i = \alpha_i^* = 0$, so we get that only support vectors influence the function. Secondly only points outside the ϵ tube will have a positive slack variable ξ or ξ^*, so then α or α^* will be equal to C. We have $\boldsymbol{\beta} = \sum_{i=1}^n (\alpha_i^* - \alpha_i) x_i$, so the regression function can be written as

$$f(x) = \beta_0 + \sum_{i=n}^{n} (\alpha_i^* - \alpha_i) \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle$$

which is the so-called Support Vector expansion of $\boldsymbol{\beta}$. Thus we need not to estimate the regression coefficients $\boldsymbol{\beta}$, even when calculating $f(x)$. The expansion of $\boldsymbol{\beta}$ is sparse in terms of x_i, as only a subset of x_i are needed to describe $\boldsymbol{\beta}$. To calculate β_0 we look at the KKT conditions. If we have an $\alpha_i \in (0, C)$, then $\xi_i = 0$ and

$$\beta_0 = y_i - \boldsymbol{\beta}^\intercal \boldsymbol{x}_i - \epsilon$$

The same can be done for an $\alpha_i^* \in (0, C)$ yielding

$$\beta_0 = y_i - \boldsymbol{\beta}^\intercal \boldsymbol{x}_i + \epsilon$$

Another way of computing β_0 is to use interior point optimization for the dual problem. Then the optimization process will return the value of β_0 as a by-product. To extend this method to non-linear relationships between input and output data we can substitute the inner product by a kernel $K(\boldsymbol{x}_i, \boldsymbol{x}) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}) \rangle$ satisfying Mercer's Theorem. The optimization problem is then to find the smoothest function in the enlarged feature space. The resulting function in the non-linear case is similarly to the linear case given by

$$f(x) = \beta_0 + \sum_{i=n}^{n} (\alpha_i^* - \alpha_i) K(\boldsymbol{x}_i, \boldsymbol{x})$$

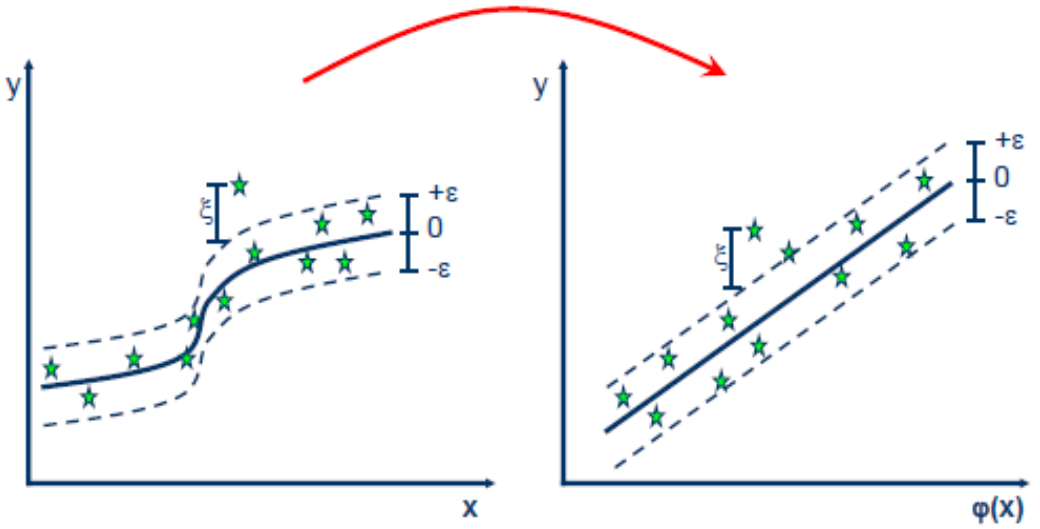


Figure 8: Example of support vector regression with transformation to a higher dimension. From Sayad (n.d.)

An important aspect of SVM/SVR is the choice of kernel. For SVR the

linear kernel only allows linear relationships between input and output, while other kernels can accommodate non-linear relationships. Chen et al. (2010) found that none of the linear, polynomial or the Gaussian kernels performed consistently better in volatility prediction. Tang et al. (2009) argue that general kernels, such as the Gaussian kernel, are not capable of capturing the volatility clusters accurately. They suggest that the wavelet kernel will be a better choice of kernel, as it handles both local characteristics and outliers well. Li (2014) was the first to apply the SVM to estimate APARCH models. The results showed that the wavelet kernel consistently outperformed the Gaussian kernel in the APARCH model. Monte Carlo simulations also showed that SVM based methods outperform QML estimation in prediction and estimation of volatility in APARCH models when the underlying data distribution is skewed t-distributed.

4 Applying SVR to APARCH

4.1 Motivation

There are a few studies that apply APARCH models in the analysis of volatility in financial data, which generally are based on maximum likelihood estimation (MLE). Olowe (2009) applied APARCH models to Naira/Dollar exchange rate volatility. Stavroyiannis (2016) used an APARCH model with the residuals following the standardized Pearson type IV distribution applied to WTI oil prices, and found improved performance compared to other residual distributions. Nugroho and Susanto compared the performance of APARCH models using normal and Student-t error distribution on daily returns of five foreign currency exchange rates to Indonesian rupiah. Many more studies apply extensions of the GARCH model, such as the GJR-GARCH and TS-GARCH models, that also fall under the APARCH model structure. The efficiency of MLE depends on the distribution of data and innovations in the dataset. MLE methods are optimal when the residual errors are normally distributed. If the distribution of innovations is not consistent with the model assumptions, the MLE can be inefficient (Engle and González-Rivera, 1991; Bollerslev and Wooldridge, 1992). To mitigate this a non-parametric model can be used. SVR is such a non-parametric method, that can be combined with APARCH models to estimate and forecast volatility. Peréz-Cruz *et al* (2003) found a better forecasting ability of the GARCH models estimated by using SVR than those estimated by using MLE when predicting the conditional volatility of stock market returns. The APARCH-SVR combination can be viewed as a semi-parametric method, as it combines the nonparametric data-driven SVR method with the parametric APARCH model structure. Using SVR we can better approximate the non-linear characteristics of the data, such as volatility clustering, excess kurtosis and leverage effects. The calculation of the Lagrange multipliers needs no a priori assumptions of the underlying data distribution. SVR is based on the structural risk minimization

of Vapnik (1992) and is thus expected to have good forecasting abilities.

4.2 Specifying input and output

We have presented the APARCH models, where h_t is the stochastic process of the conditional variance of the return series Z_t. The conditional variance can be predicted as a function of the long run unconditional variance w, the one lag squared residual Z_{t-1} and the one lag conditional variance h_{t-1}

$$Z_t = e_t\sqrt{h_t}, \quad e_t \sim IID(0,1)$$
$$h_t^{\delta/2} = w + \alpha(|Z_{t-1}| - \gamma Z_{t-1})^\delta + \beta_j h_{t-1}^{\delta/2}$$

The APARCH model is typically estimated by Maximum Likelihood when e_t are Gaussian distributed and Quasi Maximum Likelihood when the distribution is non-normal. Bollerslev and Wooldridge (1992) showed that the normal QML estimate is consistent and asymptotically normal under fairly weak regularity conditions, but the QML estimate is not asymptotically efficient under non-normality and cannot provide the best estimates for smaller sample sizes.

To use SVR to estimate APARCH models, we don not need to estimate the coefficents of the APARCH model. We must specify the input and output to the function $f(\boldsymbol{x}_t)$. The output will be the conditional variance estiamte of $h_t^{\frac{\delta}{2}}$. The input will be $\boldsymbol{x}_t = [(|Z_{t-1}| - \gamma Z_{t-1})^\delta, h_{t-1}^{\delta/2}]$ if γ is given. If γ is not specified, then the input will depend on the model selected. The power term $(|Z_{t-1}| - \gamma Z_{t-1})^\delta$ is expanded. Then for the GARCH($\delta = 2, \gamma = 0$), TS-GARCH($\delta = 1, \gamma = 0$), T-GARCH($\delta = 1$) and GJR-GARCH($\delta = 2$) we have the input vectors $\boldsymbol{x}_t = [Z_{t-1}^2, h_{t-1}]$, $\boldsymbol{x}_t = [|Z_{t-1}|, h_{t-1}^{1/2}]$,$\boldsymbol{x}_t = [|Z_{t-1}|, Z_{t-1}, h_{t-1}^{1/2}]$, and $\boldsymbol{x}_t = [Z_{t-1}^2, |Z_{t-1}|Z_{t-1}, h_{t-1}]$ respectively. h_{t-1}, which is the volatility at the previous time step is not observable. Instead we need to use an approximation. Peréz-Cruz *et al* suggested the approximation $h_t^* = \frac{1}{5}\sum_{k=0}^{4} Z_{t-k}^2$. Li (2014)

showed that this approximation results in an over-smoothing of the realized volatility, and suggested instead to use $h_t^* = \frac{1}{3}\sum_{k=0}^{3} Z_{t-k}^2$ as input variable.

4.3 Wavelet kernel

The wavelet transform methods were developed in the 1980's by the likes of Grossman and Morlet (1984), Daubechies (1988) and Mallat(1989), although the first wavelet was proposed by Haar in 1909. Common applications of wavelet transforms are data compression, such as image processing (for example the JPEG 2000 compression standard), and signal processing. Wavelet transforms are similar to Fourier transforms. Fourier transform can be viewed as a special case of continuous wavelet transform. The most important difference of wavelet transforms is that they are localized both in time and frequency, whereas Fourier transforms are localized in frequency only. Thus, wavelet transforms can be applied to signals that change over time. The orthonormal wavelet bases $\psi_{a,c} : \ a, c \in R$ are generated by translations and dilations of a single function called the mother wavelet $\psi(t) \in L^2(R)$ defined by

$$\psi_{a,c}(x) = \frac{1}{\sqrt{a}}\psi(\frac{x-c}{a})$$

Here a is a scale parameter, and c a location parameter. Smaller values of a will compress the wavelet in the time dimension and increase the frequency, while larger values will result in lower frequency and wider time width. In wavelet analysis a is called the dilation factor, and c is called the translation factor. The mother wavelet is a zero-mean and normalized function, meaning

$$\int_{-\infty}^{\infty} \psi(t)dt = 0$$

$$||\psi(t)||^2 = 1$$

also the mother wavelet has to satisfy the admissibility condition

$$W_\psi = \int_0^\infty \frac{|\Psi(\omega)|^2}{|\omega|} d\omega < \infty$$

where $\Psi(\omega)$ is the Fourier transform of $\psi(x)$. For a signal $f(x)$, the wavelet transform is

$$W_{a,c}(f) = \langle f(x), \psi_{a,c}(x) \rangle = \int_{-\infty}^{\infty} f(x) \psi^*_{a,c}(x) dx$$

Given the admissibility condition, the function f can be reconstructed by the inverse wavelet transform

$$f(x) = \frac{1}{W_\psi} \int_{-\infty}^{\infty} \int_0^\infty W_{a,c}(f) \psi_{a,c} \frac{dadc}{a^2}$$

or using finite terms, by the approximation

$$\hat{f}(x) = \sum_{i=1}^{n} W_i \psi_{a_i,c_i}(x)$$

To apply the wavelet transform to SVM we use a multi-dimensional wavelet function defined by

$$\psi_d(\boldsymbol{x}) = \prod_{j=1}^{d} \psi(x)$$

Zhang *et al.* (2004) proposed two admissible, symmetric and positive definite, kernel functions based on wavelets to SVM that satisfy Mercer's condition. The wavelet kernels are multidimensional wavelet functions that can approximate arbitrary nonlinear functions. Zhang et al. combined the wavelet theory and support vector machines and showed that the wavelet

kernel can achieve more accurate approximation for nonlinear functions than other kernels. The dot product kernel is defined by

$$K(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{d} \psi(\frac{x_i - c_i}{a})\psi(\frac{y_i - c_i}{a})$$

and the translation invariant wavelet kernel, that satisfy the translation invariant kernel theorem, is

$$K(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{d} \psi(\frac{x_i - y_i}{a})$$

where $\boldsymbol{x}, \boldsymbol{y} \in R^d$, a and c denotes the dilation and translation and $\psi(x)$ is a mother wavelet. The translation invariant kernel theorem states that a translation invariant kernel, $K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{x} - \boldsymbol{y})$, is an admissible support vector kernel if and only if the Fourier transform

$$F[K](\omega) = (2\pi)^{-d/2} \int_{R^d} exp(-j(\omega \cdot \boldsymbol{x}))K(\boldsymbol{x})d\boldsymbol{x}$$

is non-negative. Proof of the theorem can be found in Schölkopf and Smola (2004), and proof that the translation invariant wavelet kernel does indeed satisfy the condition of the theorem can be found in the appendix of Zhang *et al* (2004). We can also easily show that the dot-product kernel is admissible. For some function $f(x) \in L^2$

$$
\begin{aligned}
&\int\int f(x)K(x,y)f(y)dxdy \\
&= \int\int f(x)\prod_{i=1}^{d}\psi(\frac{x_i - c_i}{a})\psi(\frac{y_i - c_i}{a})f(y)dxdy \\
&= \int f(x)\prod_{i=1}^{d}\psi(\frac{x_i - c_i}{a})dx \int f(y)\prod_{i=1}^{d}\psi(\frac{y_i - c_i}{a})dy \\
&= \left(\int f(x)\prod_{i=1}^{d}\psi(\frac{x_i - c_i}{a})dx\right)^2 \geq 0
\end{aligned}
$$

A choice of mother wavlet proposed by Zhang was the Morlet wavelet

$$\psi(x) = cos(1.75x)exp(-\frac{x^2}{2})$$

Combining the Morlet mother wavelet with the translation invariant wavelet kernel we obtain the following kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{d}\psi(\frac{x_i - y_i}{a}) = \prod_{i=1}^{d} cos(1.75\frac{x_i - y_i}{a})exp(-\frac{||x_i - y_i||^2}{2a^2})$$

This is an admissible support vector kernel, proved by Zhang *et al.* The estimate function for wavelet SVM is the approximation

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\prod_{j=1}^{d}\psi(\frac{x^j - x_i^j}{a})$$

where the x_i^j denotes the jth component of the ith training data-point. Zhang *et al* applied the wavelet kernel to three experiments, and found improved performance when compared to the Gaussian kernel. Their paper concluded by noting that the wavelet kernel has better approximation than the Gaussian

kernel and faster training speed. They attribute this to the fact that the wavelet kernel is orthonormal (or orthonormal approximately), whereas the Gaussian kernel is not and thus the Gaussian kernel is correlative.

5 Empirical Studies

5.1 Setup

We will use maximum likelihood and SVR in APARCH models to estimate and forecast conditional variance for a dataset of financial returns $(t_1, x_1), ..., (t_n, x_n)$, where x_i is the return at time point t_i. The performance will be measured by mean squared error (MSE) $\frac{1}{n}\sum_{i=1}^{n}(h'_t - h^*_t)^2$ and mean absolute error (MAE) $\frac{1}{n}\sum_{i=1}^{n}|h'_t - h^*_t|$, where h'_t is the conditional volatility from our models and $h^*_t = \sum_{k=0}^{3} Z^2_{t-k}$ is an approximation of h_t using the return series Z_t. The true value of h_t is of course not observable, we can only observe realized volatility. The MSE and MAE will be calculated for both training and test data. The training data are used to estimate the Lagrange multipliers of the SVR and the parameter vector $\boldsymbol{\theta} = (\alpha, \beta, \delta, \gamma, w)$ from the QML in the APARCH model. The conditional variance h'_t from training data is the estimated volatility, while for test data it is the forecasted volatility. The test data is not used to fit the model and the model can be used in real time to forecast volatility. Test data performance is the most important measure, as it tells whether the model is useful out of sample or just a good fit in sample. Each of the datasets are split up into two or three periods. Each period is then divided into a training dataset and a test dataset, where 3/4 of the period is used as training data and 1/4 as test data. The values of ϵ and C in the SVR models are chosen by grid search. Grid search is a brute-force method for hyperparameter selection. A subset of values is manually chosen for each hyperparameter, creating a grid of possible combinations. Then every combination of hyperparameter values is tested using cross validation to find the best one. Performance is measured by mean squared error. Since we only have two hyperparameters to tune this is a viable method for hyperparameter selection. To make sure the grid is reasonable, different grids have been tested so that the values of C and ϵ are somewhere in the middle of the grid. The exact value of C and ϵ may vary even with constant grid, depending on how

the cross validation divides the data. The value of a in the Wavelet kernel is chosen to be 2 so that

$$K(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{n} cos(1.75\frac{x_i - y_i}{2})exp(-\frac{||x_i - y_i||^2}{8})$$

We compare the performance of the QML method with SVR. Two different kernels are used, the Gaussian and the wavelet. Four different APARCH models are used, the GARCH, GJR, TARCH and TSGARCH. The datasets used are the S&P 500 stock index, high yield corporate credit spreads and Nordic el-spot prices. Two additional kernels, the linear and the polynomial, are also used with performance listed in the appendix.

5.2 S&P 500

The S&P 500 is a stock market index that measures the stock returns of 500 of the largest publicly listed companies in the United States. The index is one of the most followed equity indices in the world, and a common benchmark for stock returns in general. Mutual funds and other money managers often quote their results in relation to the performance of the S&P 500. The index also serves as a basis for investable products, such as exchange traded funds (ETFs), including three of the four largest ETFs in the world measured by assets under management, mutual funds, and futures, options and other derivative contracts. The total market capitalization, the number of shares outstanding multiplied by market price per share for each index constituent, of the index is USD 28.5 trillion, and there is over USD 11.2 trillion indexed or benchmarked to the index (As of October 30. 2020 spdji.com). The index is maintained by S&P Dow Jones Indices, a joint venture between S&P Global Inc. and CME Group Inc. The components of the index are determined by a committee. The criteria for inclusion includes a market capitalization greater than USD 8.2 billion, with a public float greater than USD 4.1 billion. The

company must be profitable in the most recent quarter, and in total over the previous four quarters. The trading volume of the stock must be minimum 250,000 shares in each of the last six months. The stock must be common equity of a U.S. company or Real Estate Investment Trust (REIT) listed on either the New York Stock Exchange or Nasdaq.

The data in this section is the daily closing price of the S&P 500, from January 3. 2006 to December 30. 2020. The data source is Yahoo! Finance, gathered by using the R-package QuantMod. The centered daily return is $u_t = r_t - \mu_{r_t}$, where $r_t = 100 * (exp(log(Y_t - Y_{t-1})) - 1)$ and Y_t is the closing price at time t. The figures below feature the price series and return series for the S&P 500. We see that equity prices have been generally increasing in the last 15 years, with two major drawdowns, during the financial crisis of 2008-2009 and the Covid-19 outbreak in early 2020. There are also a few smaller drawdowns, most notably during the European debt crisis in 2011, the slowdown in China and other emerging markets and plummeting oil prices in 2015-2016 and in 2018 following the Federal Reserve tightening monetary policy. The skewness of the return series is measured at -0.2690744, indicating that negative returns tend to be larger than positive returns. The excess kurtosis is measured at 13.25763, meaning we have heavy tails.

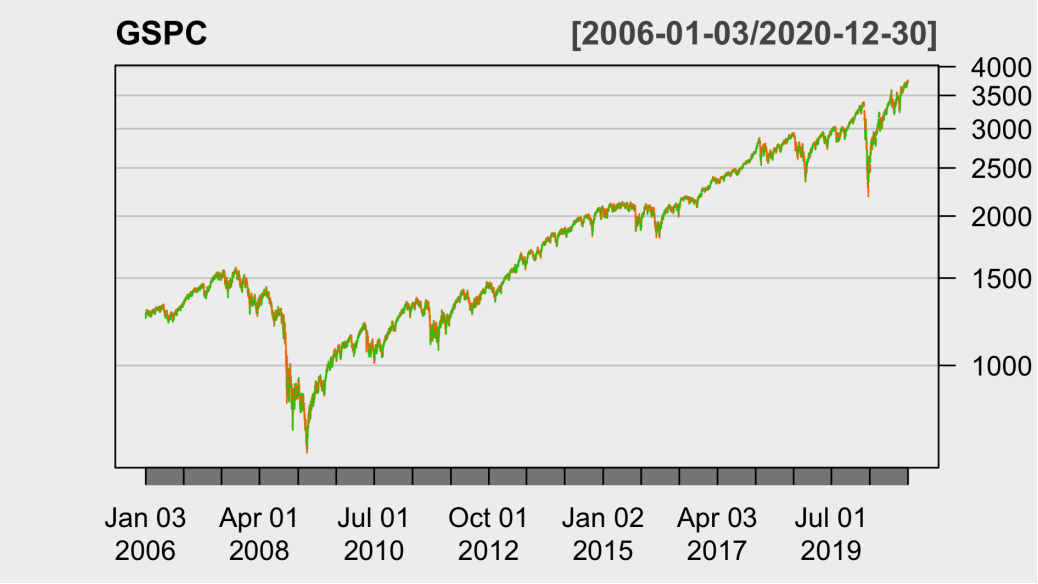


Figure 9: S&P 500 price level from January 3. 2006 - December 30. 2020

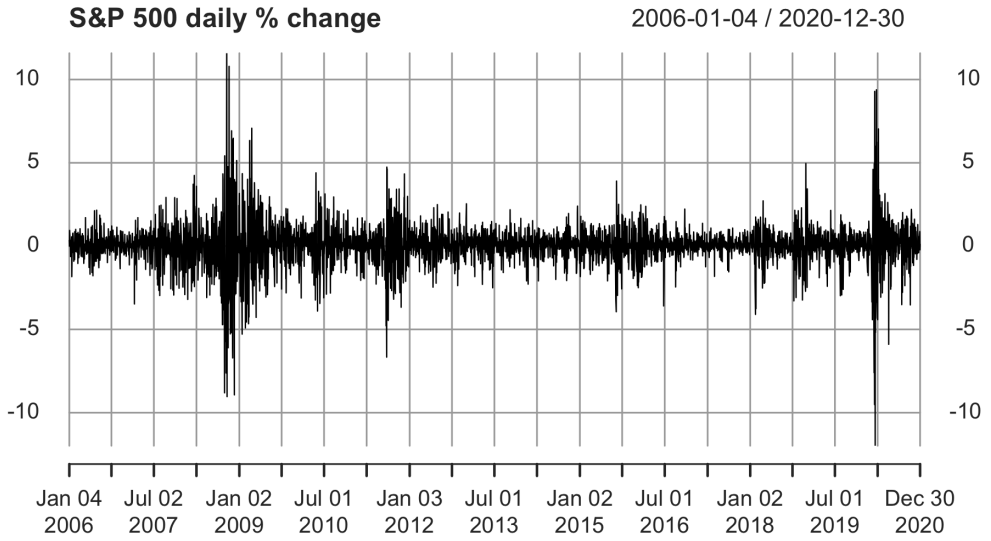
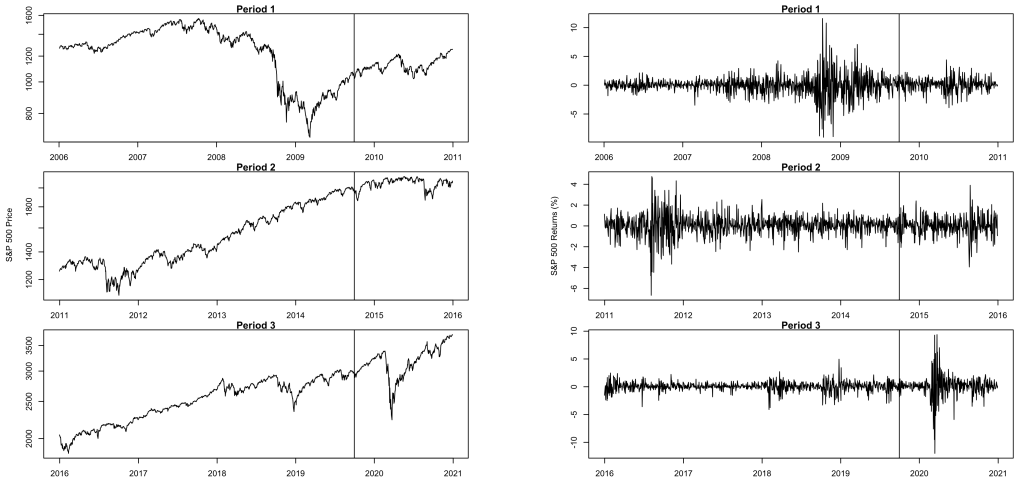


Figure 10: S&P 500 daily return from January 3. 2005 - December 30. 2020

The dataset is divided into three time periods; 2006-2010, 2011-2015

and 2016-2020. The plot below shows the S&P 500 price level and realized volatility for the three periods, with training data on the left side of the vertical line and test data on the right. The first period contains the financial crisis, with very high volatility, as training data. The test period volatility is lower. The third period is the opposite with mostly very low volatility in the training data while the test data contains the Covid-19 crash. The second period has much more balanced volatility in the training and test data.



(a) S&P 500 price level (b) S&P 500 returns

Figure 11: S&P 500 daily prices and returns for the three subperiods

As preliminaries, we check characteristics related to the centered return series u_t. The number of observations is denoted by N. The Shapiro-Wilk test (Shapiro and Wilk, 1965) is used to test for normality of the centered returns u_t, with the null hypothesis that the dataset is normally distributed. The Box-Ljung test (Ljung and Box, 1978) is used to test for correlations of u^2 with lag one and five, corresponding to one day and one trading week. The Lagrange Multiplier test (Engle, 1982) is used to test for ARCH-effects, with the null hypothesis of no ARCH effect. The test should also have power against GARCH effects (Bollerslev, 1986). Here the test is performed at lag

10 for the LM test. The p-value of the test results are shown in the table below.

	Period 1	Period 2	Period 3
N	1305	1305	1305
Shapiro-Wilk test	< 2.2e-16	< 2.2e-16	< 2.2e-16
Box-Ljung test lag 1	1.314e-10	< 2.2e-16	< 2.2e-16
Box-Ljung test lag 5	< 2.2e-16	< 2.2e-16	< 2.2e-16
LM test lag 10	< 2.2e-16	< 2.2e-16	< 2.2e-16

Table 1: Preliminaries

All the test results are extremly significant. There is highly significant non-normality as a result of the heavy tails and conditional heteroscedasticity. There is also significant evidence of correlations in u^2 for both lag one and lag five. Since the LM test is also highly significant, we conclude that GARCH-type models are applicable. For the GJR and TARCH models we need to estimate the value of γ in the QML estimated models. For all time periods there is highly significant positive value of γ, for both the GJR and TARCH models. This means that volatility increases more with large negative returns than positive returns, and that leverage effects should be included in the model. For the support vector regression models we are interested in the number of support vectors as a measure of computational efficiency. The wavelet kernel generally uses fewer support vectors than the Gaussian kernel. Fewer support vectors can also help to reduce errors in the test dataset as the variance of the model decrease.

	Model	Estimate	Std. Error	t-value	p-value
Period 1	GJR	1.000000	0.331784	3.014	0.00258
	TARCH	1.000000	0.016259	61.504	< 2e-16
Period 2	GJR	1.000000	0.272198	3.674	0.000239
	TARCH	1.000000	0.010515	95.099	< 2e-16
Period 3	GJR	0.458995	0.131427	3.492	0.000479
	TARCH	1.00000	0.016985	58.875	< 2e-16

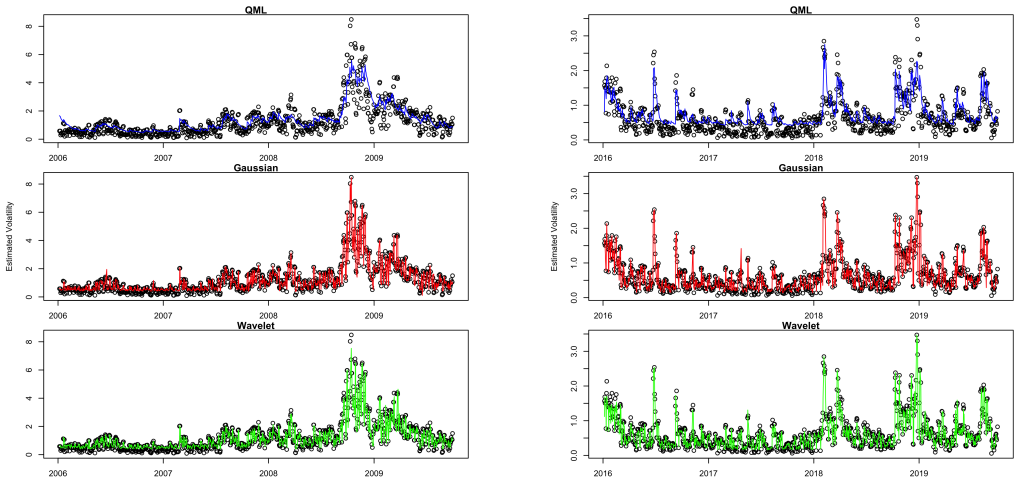
Table 2: Estimate of γ

	Kernel	GARCH	GJR	TARCH	TSGARCH
Period 1	Gaussian	652	663	828	609
	Wavelet	576	574	800	578
Period 2	Gaussian	734	755	683	701
	Wavelet	721	714	680	676
Period 3	Gaussian	924	921	923	844
	Wavelet	931	910	922	845

Table 3: Number of support vectors

	Period 1			Period 2			Period 3		
MSE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	16.81715	1.32472	8.71106	1.64263	0.80463	1.01889	0.53044	0.08023	0.31761
GJR	15.69177	1.22932	8.14722	1.25389	0.30887	0.85007	0.49812	0.06708	0.24358
TARCH	15.40920	4.32664	10.55776	1.64690	0.74614	0.97648	0.55091	0.17364	0.34106
TSGARCH	18.87404	3.12808	9.79917	2.08243	0.68665	0.99345	0.65626	0.16714	0.38384
MAE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	1.66402	0.37845	0.99268	0.51935	0.26937	0.37978	0.36890	0.09394	0.23745
GJR	1.59100	0.35872	0.91649	0.49654	0.21399	0.34225	0.37197	0.07673	0.21045
TARCH	1.63374	0.63234	1.00173	0.52440	0.30809	0.35291	0.39436	0.15159	0.23384
TSGARCH	1.73324	0.67384	1.08036	0.55076	0.31391	0.37471	0.40641	0.16174	0.25334

Table 4: S&P 500 Training Error (MSE & MAE)



(a) Period 1 (b) Period 3

Figure 12: S&P 500 daily conditional standard deviation in percent, approximation $\sqrt{h_t^*}$ (dots) and GJR model estimate $\sqrt{h_t'}$ (training data)

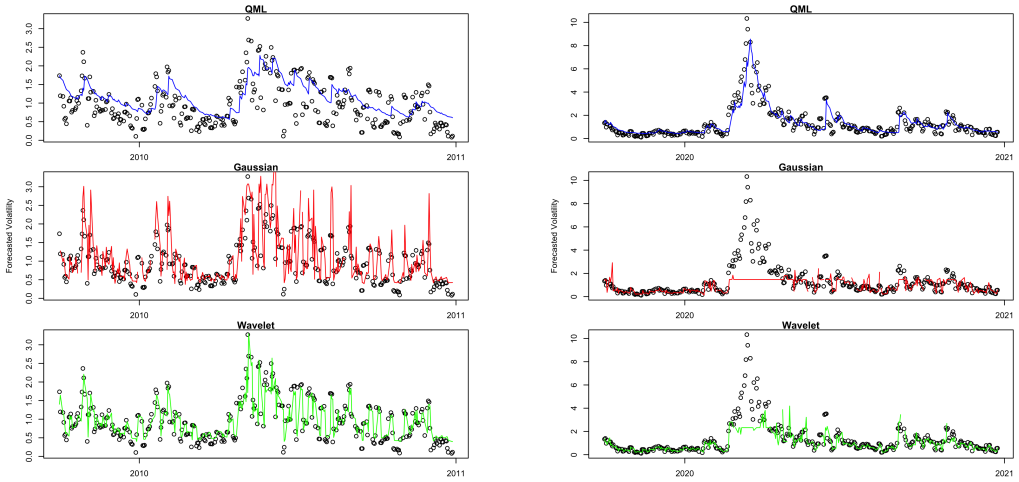
The training set MSE and MAE for the different models applied to the S&P 500 dataset is provided above. The general tendency is for the QML models to perform clearly worst, and for the gaussian kernel to slightly outperform the wavelet kernel. It is difficult determine which APARCH model works best, although the TSGARCH performs worst in every period for QML based models. The plots above shows square root of h_t^* as dots and the square root of h_t' in the GJR models as lines. From the plots we observe that the SVR models are much more flexible than the QML models. The SVR models are therefore able to capture a majority of the volatility spikes, as well as to quicker adjust to lower volatility.

	Period 1			Period 2			Period 3		
MSE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	1.51006	3.77668	1.07864	1.09265	0.87601	0.55606	51.70217	124.5159	113.6029
GJR	1.39698	4.20442	1.14557	0.79828	0.94426	0.42858	48.51570	125.3363	111.9718
TARCH	1.48663	1.65659	1.19918	0.92261	0.56889	0.49906	50.11123	124.4605	118.5207
TSGARCH	1.52570	1.45023	1.10210	1.25320	0.75262	0.58780	57.14117	122.1579	108.0572
MAE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	0.88765	0.98843	0.58218	0.57520	0.44372	0.38258	2.25864	3.39925	2.89213
GJR	0.85132	1.05469	0.59407	0.49856	0.47754	0.36921	2.11129	3.35681	2.93830
TARCH	0.86606	0.66888	0.58669	0.52932	0.40217	0.38072	2.15640	3.26802	3.02260
TSGARCH	0.89269	0.64599	0.57954	0.61574	0.42473	0.38952	2.34104	3.16420	2.71566

Table 5: S&P 500 Test Error (MSE & MAE)

The above tables show the test set MSE and MAE for the S&P 500 dataset. The first two periods are clearly best estimated by the wavelet kernel. The final period contains the relatively stable period between 2016 and 2019 as training data, including 2017 which was one of the lowest realized volatility years ever. The test data however contains the extreme volatility of the financial markets during the Covid-19 outbreak in 2020. There is a major structure break that is not captured by neither the Gaussian nor the Wavelet kernel models. The result is terrible performance measured by MSE and MAE. The result for the QML model is also bad, but it captures at least some of the extreme volatility. We can therefore infer that it is important to

train models on data that contains most of the possible range of volatility. This is a common problem in finance, where models are trained on data that are not representable of what can happen, only what has happened in some limited period of time. Alas, the true fat-tailedness of stock returns is not present in training data, which can lead to "blow-ups" if a far-left tail event breaches risk measures (such as Value at Risk). From the plots we can see that the Wavelet kernel model performs very well in period one, while the Gaussian kernel model seems overreactive. The first period has extreme volatility in the training data, during the Great Financial Crisis, while the test data has relatively low volatility (below 3% daily volatility). In period three the volatility reaches 10% daily change and both SVR models fail, while the QML model performs reasonably well. The plot for period two can be found in the appendix.



(a) Period 1 (b) Period 3

Figure 13: S&P 500 daily conditional standard deviation in percent approximation and GJR model forecast (test data)

5.3 Credit Spreads

Sale of corporate bonds are one of the main venues for corporations to raise debt capital. Investors purchasing the bonds in return typically receives annual, semi-annual or quarterly interest payments. At the maturity of the bond, the corporation pays the debtholders the principal of the bond. Corporate bonds are partitioned by corporate credit rating and time to maturity. S&P Global Ratings (S&P), a division of S&P Global Inc., is a financial services company that, among other services, assigns credit ratings to various debt securities. Credit ratings rate the debtor's ability to pay back interest and principal, as well as the likelihood of default. S&P's ratings range from AAA as the safest to D for a company that has already failed to pay its financial obligations at due date. Ratings from AAA to BBB- are considered Investment grade. Bonds rated BB+ and lower are considered non-investment grade and are often referred to as high-yield or junk bonds. These bonds will carry substantial risks.

Credit spreads are defined as the difference between the interest yield of a uncertain (corporate) bond, and debt issued by the United States Treasury with comparable maturity. Credit spreads of bonds in aggregate are influenced or correlated by a number of factors. Factors of significance includes interest rates, interest rate volatility, yield curve slope, bond liquidity, exchange rates, stock market returns and stock market volatility (Clark *et al* 2018). Bonds with lower ratings will have larger interest payments, thus larger credit spreads, to compensate for the increased risk of default. Yields and credit spreads tends to increase for longer times to maturity. Higher rated and longer dated bonds also tend to exhibit larger credit spread volatility. The credit spread is an important measure of the availability of funding for corporations. Tight spreads indicate that investor appetite for bonds is high, and funding is readily available at good terms for corporations. Spreads are usually high in economic downturns, as investors demand more yield to compensate for

perceived increased risk. Thus, funding will be less available, and at a higher cost, for corporations, which might add more pressure on the economy.

In this section we examine the ICE BofA US High Yield Index Option-Adjusted Spread. The data is gathered from the website of the United States Federal Reserve Bank of St. Louis' Federal Reserve Economic Data (FRED) https://fred.stlouisfed.org/categories/32348. The frequency is daily, and the measure is the percentage point spread above the yield on US treasuries for US corporate high yield bonds. Each security must have greater than 1 year of remaining maturity, a fixed coupon schedule, and a minimum amount outstanding of $100 million. The time period examined is 2006-2020, and the dataset is divided into the same three five-year periods as S&P 500 dataset. We can clearly see the financial crisis, where high yield credit spreads rose to over 20%. We also got a large spike in the credit spreads following the Covid-19 virus outbreak in 2020. Credit spreads were also fairly high in the period 2014-2016, as many energy companies struggled with falling oil prices. The return series is denominated by the basis points (Bps, 100Bps=1%) change in credit spreads. There are clearly volatility clusters at the periods of elevated spreads. This includes in particular the great financial crisis of 2008-2009, where high yield bonds were trading at prices yielding more than 20% more than Treasury bonds. Bonds were never as depressed in the Covid-19 crash, but the volatility almost as high as change in yield happened very fast (in both directions).

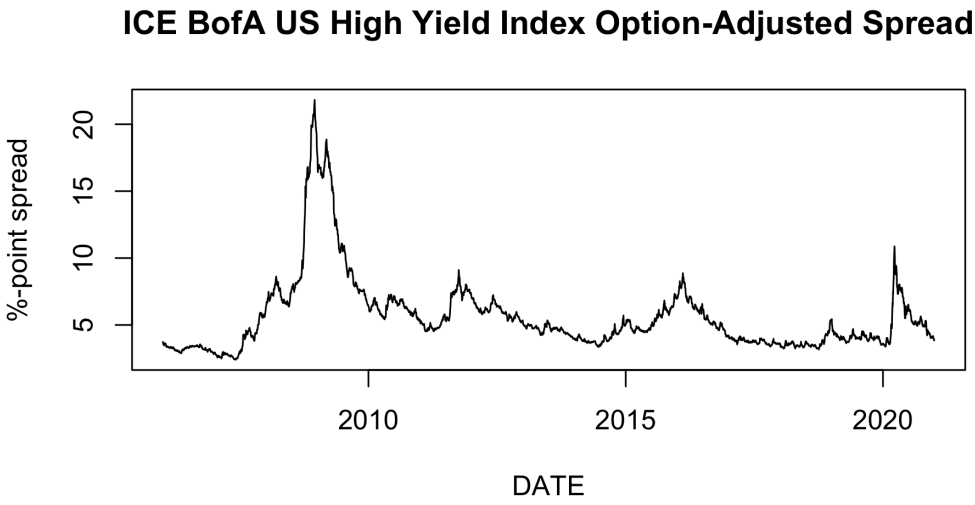


Figure 14: High Yield spreads from 2006 to 2020

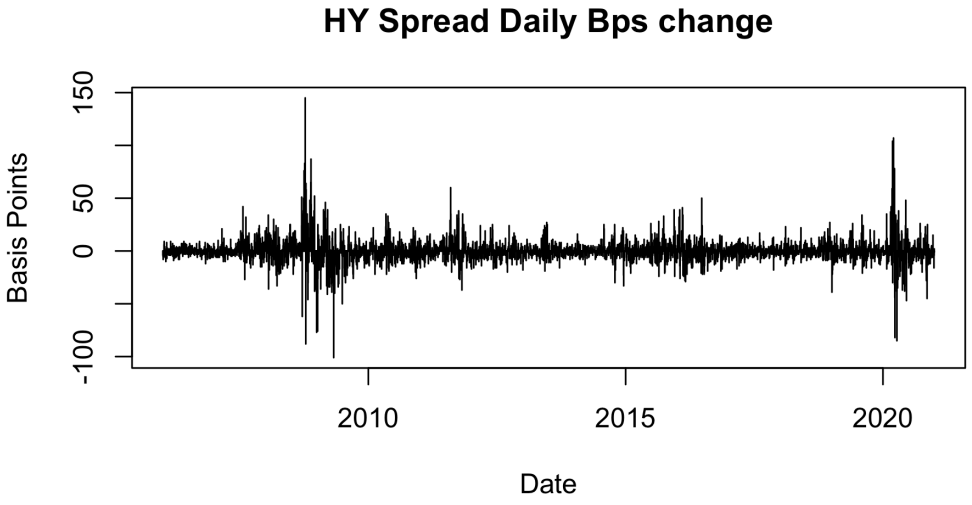
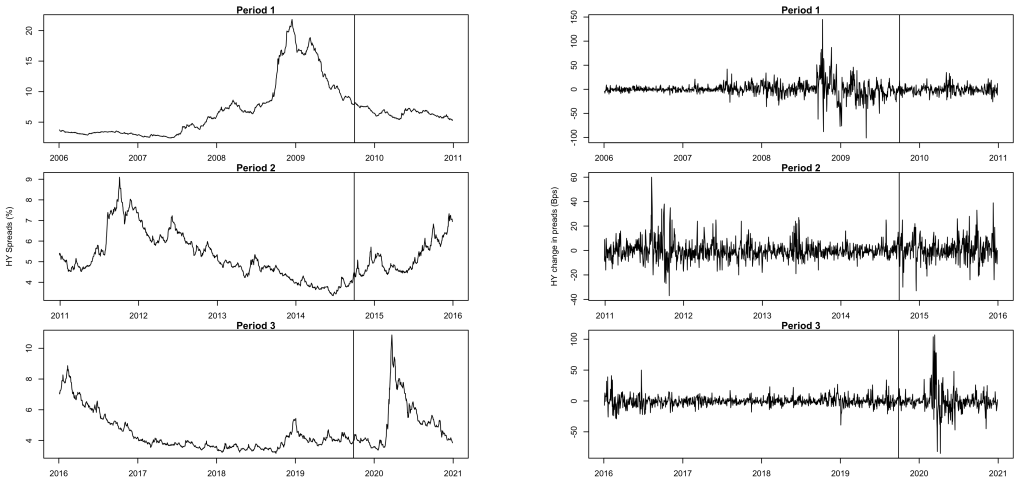


Figure 15: High Yield change in spreads from 2006 to 2020

The dataset is again divided into three time periods; 2006-2010, 2011-2015

and 2016-2020. The plot below shows the High Yield credit spreads level and the change daily change in credit spreads for the three periods. The vertical line divides the test and the training data. The first period contains the financial crisis, with extremely high volatility, as training data. The test data mostly has compressing spreads from the high levels of financial crisis and lower volatility. The third period is the opposite. Credit spreads are generally compressing and very low in the trainin data, with muted volatility. The test data contains the Covid-19 crash, with extreme volatility. The credit spreads and volatility rose sharply with the uncertainty around the Covid-19 outbreak, but quickly fell down again after the Federal Reserve announced that it was buying large quantities of bonds and other credit securities. The second period has a period of relatively high volatility during the European debt in 2011 in the training data. In the test data the end of 2015 also experienced some volatility as many oil fracking companies with large debt loads struggled with falling oil prices. Overall, the volatility in the training and test data is fairly even, with no extremes.



(a) HY spred level (b) HY spread change

Figure 16: High Yield spreads and spread changes for the three subperiods

	Period 1	Period 2	Period 3
N	1234	1234	1234
Shapiro-Wilk test	< 2.2e-16	< 2.2e-16	< 2.2e-16
Box-Ljung test lag 1	7.139e-14	< 2.2e-16	< 5.44e-15
Box-Ljung test lag 5	< 2.2e-16	< 2.2e-16	< 2.2e-16
LM test lag 10	< 2.2e-16	< 2.2e-16	< 2.2e-16

Table 6: Preliminaries

This data set also has significant test results for all periods, so GARCH type models are applicable. There is clearly non-normality, correlation in the second order and ARCH effects. The estimates of γ are also significant, suggesting leverage effects. The sign of the γ coefficient is negative for this data set. This means that credit spreads volatility increases when credit spreads are increasing. This makes sense as credit spreads increase in times of financial distress and liquidity constraints. The spread is negatively correlated with the stock market returns studied in the previous section. The number of support vectors is again generally lower for the wavelet based models.

	Model	Estimate	Std. Error	t-value	p-value
Period 1	GJR	-0.15118	0.04959	-3.048	0.0023
	TARCH	-0.23039	0.05751	-4.006	6.16e-05
Period 2	GJR	-0.19577	0.06836	-2.864	0.00418
	TARCH	-0.30285	0.07916	-3.826	0.00013
Period 3	GJR	-0.29987	0.07707	-3.891	9.98e-05
	TARCH	-0.59888	0.10851	-5.519	3.41e-08

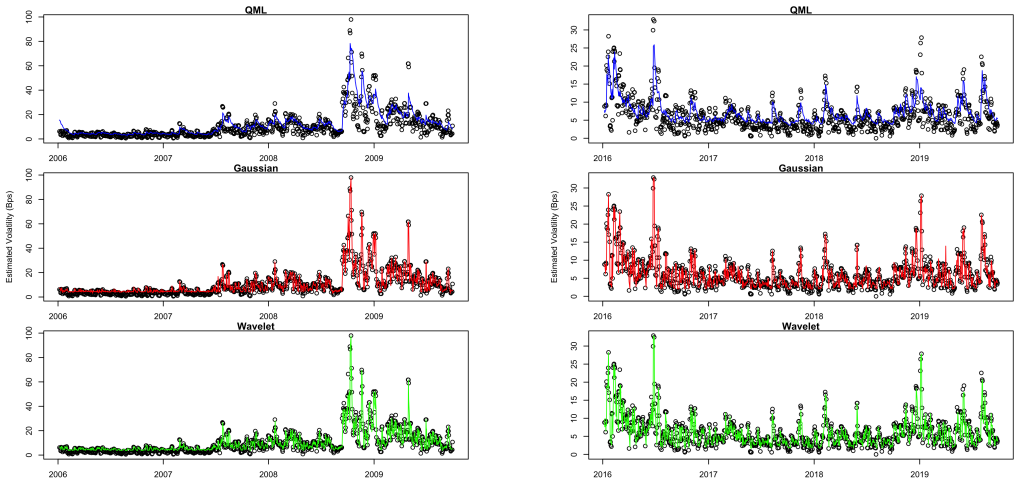
Table 7: Estimate of γ

	Kernel	GARCH	GJR	TARCH	TSGARCH
Period 1	Gaussian	437	550	715	717
	Wavelet	365	467	682	679
Period 2	Gaussian	435	439	840	845
	Wavelet	408	406	843	852
Period 3	Gaussian	900	900	946	944
	Wavelet	895	880	953	951

Table 8: Number of support vectors

	Period 1			Period 2			Period 3		
MSE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	218496.5	4360.792	101785.6	8101.01	810.05	3674.28	4769.59	1910.88	2825.17
GJR	239002.6	3730.750	100955.9	7748.41	2697.80	4269.16	5056.96	1303.05	2703.52
TARCH	255528.7	27578.188	106003.5	8402.40	1509.80	3930.25	5189.05	1876.40	2315.69
TSGARCH	243880.8	25766.276	112137.5	8972.59	1362.80	4007.75	4687.67	1913.56	2497.67
MAE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	163.382	35.599	88.823	38.199	14.907	22.732	34.410	12.668	20.636
GJR	169.825	22.518	78.647	38.762	17.371	23.891	35.766	11.316	19.706
TARCH	174.299	44.602	77.134	38.880	16.798	21.621	35.238	15.170	18.891
TSGARCH	169.663	45.899	84.442	38.293	16.365	22.584	33.670	15.778	20.060

Table 9: HY Training Error (MSE & MSE)



(a) Period 1 (b) Period 3

Figure 17: HY spread daily conditional standard deviation bps change, approximation and GJR model estimate (training data)

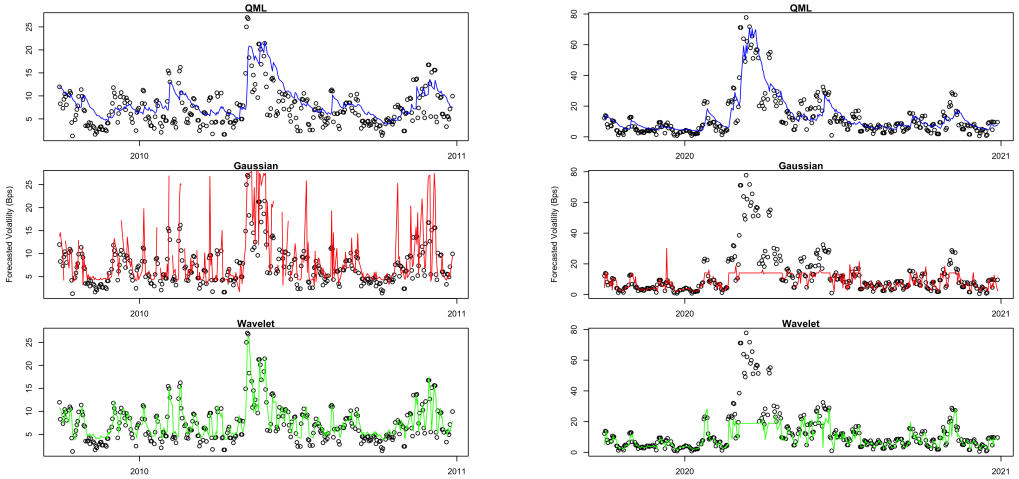
For the training data the MSE and MAE are highest for the QML method and lowest for the Gaussian kernel method. From the plots we observe that the kernel models capture almost every single volatility spike, while the QML model generally underreacts to the spikes. The kernel models also quickly come back down after a spike, while the QML model gradually moves lower. There is a large difference in the magnitude of volatility for the two periods

shown in the plot. The first period has almost 100Bps daily change in spreads during the financial crisis, while the third period barely reaches 30Bps daily change.

	Period 1			Period 2			Period 3		
MSE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	6887.1	18167.9	2886.9	11293.4	8083.5	6632.0	273732.0	777721.1	710181.6
GJR	7410.6	30890.7	3062.9	9879.7	8518.1	5004.1	267752.7	784100.4	716217.9
TARCH	8352.2	25266.9	3165.1	9948.2	6558.1	6148.3	325126.4	769984.0	760646.6
TSGARCH	7732.9	13743.4	3303.7	11504.0	7320.4	5338.1	372827.9	770903.7	727815.7
MAE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	51.026	66.619	33.596	63.014	45.243	39.156	202.844	296.519	270.456
GJR	53.398	83.194	29.418	57.963	46.138	36.280	207.448	302.060	269.353
TARCH	56.863	44.926	28.967	58.177	40.543	38.104	227.272	285.680	281.355
TSGARCH	54.951	39.405	29.119	63.228	41.588	36.087	226.918	287.197	272.819

Table 10: HY Test Error (MSE & MAE)

For the test data we again observe that the wavelet kernel method produces the lowest MSE and MAE values. The time periods studied in this dataset are the same as those in the S&P 500 dataset. We therefore get the same problem for the third period. The training data contains mostly compressed spreads and low volatility, while the test data contains the extreme volatility of February and March of 2020. For the first time period it is the opposite, with extreme volatility in the training data and low volatility in the test data. The SVR based methods are again unable to capture any of the extreme volatility of March 2020, while the QML method captures some of it. It is not possible to determine which APARCH model performs better on this dataset for neither QML nor SVR. We can see that the Gaussian kernel model is way too reactive in the first period, likely as a result of the extreme volatility in the training data. The GJR model showed in the plot seems to be the worst performing on the first time period for this kernel.



(a) Period 1 (b) Period 3

Figure 18: HY spread daily conditional standard deviation bps change, approximation and GJR model forecast (test data)

5.4 Nordic Electricity Prices

Nord Pool runs the leading power market in Europe, offering both day-ahead and intraday markets to its customers. The day-ahead market is the main arena for trading power, and the intraday market supplements the day-ahead market and helps secure balance between supply and demand. Market participants express their bids and offers in terms of prices and quantities. The electric power markets are known to be among the most volatile markets. Power supply cannot easily be stored like other commodities can, and there needs to be almost perfect balance between production and consumption at all times. Power prices are determined by supply and demand. The supply is often dependent on weather, especially for renewable energy sources such as wind and solar. The demand is also affected by the weather, as very low temperatures require more heating, and very warm weather require air conditioning. Demand tends to be inelastic to price in the short term, so that large price moves are necessary to incentivize demand to adjust to supply.

Integrating power markets across borders can help secure power supply and stabilize prices as oversupply in one region can be offset by undersupply in another. Integrated power markets also contribute to lower overall power prices and optimal division of resources. The price of electric power is also dependent on the price of other energy commodities like natural gas, heating oil and coal.

Norwegian power prices are generally more stable than many other places. This is a result of the ability regulate hydroelectric supply to match demand on a continuous basis. Thermal power plants, which can have heat sources such as nuclear, coal, gas, biofuels, solar, waste or biofuels, are more costly to regulate. Large scale thermal plants are often synonymous with oligopolistic supply side, which can further reduce flexibility of supply and increase price volatility. The ability to import power in the winter when heating demand is high and reservoir inflow is low, and to export in the summer when heating demand is low and reservoir inflows are high further smooths prices.

The prices of electric power are of extreme importance for many companies in industries such as heavy engineering and commodity refinement, for example aluminum extraction and smelting, where electricity is a large part of the input cost. Risk management of energy commodities and electricity is a major issue for these types of operations, as it can seriously affect its competitiveness, viability and profitability. Energy market participants, like the large producers, large energy consuming companies, electricity resellers and traders, need volatility forecasts for effective investment, hedging risk and arbitrage strategies. It has been observed that price forecasting errors are still high from risk management perspective (Aggarwal *et al* 2009). The history of electricity markets is relatively short and large differences in price developments exist in different power markets. Aggarwal *et al* found no systematic evidence of out-performance of one model over the other models on a consistent basis. Some research has been done applying GARCH type models

to electric power prices. Garcia *et al* (2005) used GARCH methodology to forecast hourly prices in the deregulated electricity markets of Spain and California. The GARCH model improved forecasting performance compared to an ARIMA model and incorporating demand as an explainatory variable further improved performance. Cifter (2013) used a Markov-switching GARCH model to forecast volatility in Nordic power prices. MS-GARCH models increased the predictive performance of volatility models at day-head and longer time horizon forecasting compared to standard GARCH and GJR models, indicating that the volatility is regime dependent. Liu *et al* (2013) used ARMA-GARCH and ARMA-GARCH-M models to predict hourly electricity prices in New England. dos Santos Coelho *et al* (2011) used RBF neural network methods with GARCH errors to forecast hour-ahead Spanish electricity pool prices. Their method increased forecasting performance compared to benchmark methods (AR, MA, ARMA, ARMA-GARCH). One, two and three days ahead forecasts were also accurate.

The data in this part is gathered from Nord Pool at https://www.nordpoolgroup.com/historical-market-data/. We study the Oslo daily elspot prices from the start of 2015 to the end of 2020, with prices denoted in NOK/MWh. Our goal is to measure the performance of the APARCH models in day ahead volatility forecasting. The power price data is more "spiky" in the short term than the previous datasets, and overall volatility is much higher. Some of the increases are in excess of 100%, and some of the drops are larger than 50%. Three of the drops of more than 50% are directly following a 100% or more spike in prices. There is again a clear visual support of volatility clustering. Another interesting property of this data is the seasonality. Prices are generally increasing during fall and early winter, and falling from spring until early summertime. 2020 had especially low prices during the summer, as a snowy winter and late spring filled hydropower reservoirs close to historic maximum.

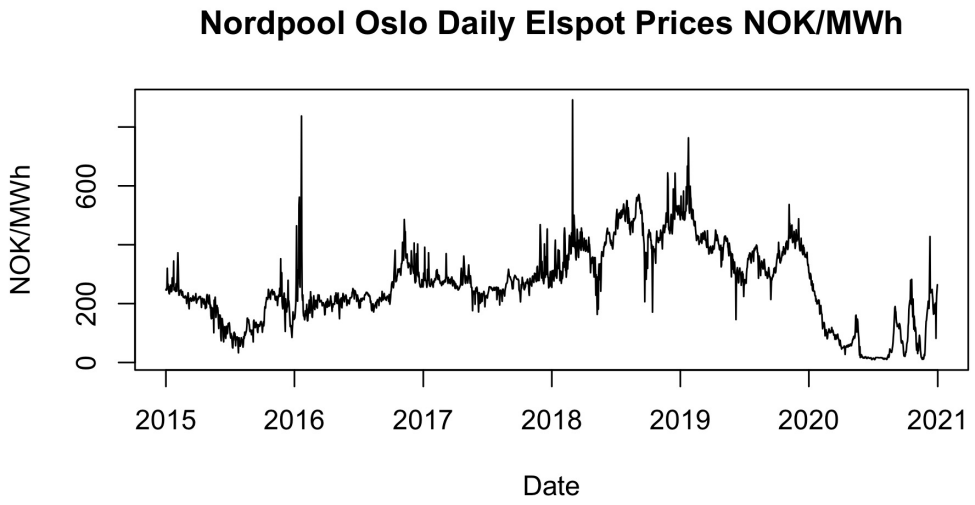


Figure 19: Oslo daily elspot prices from 2015 to 2020

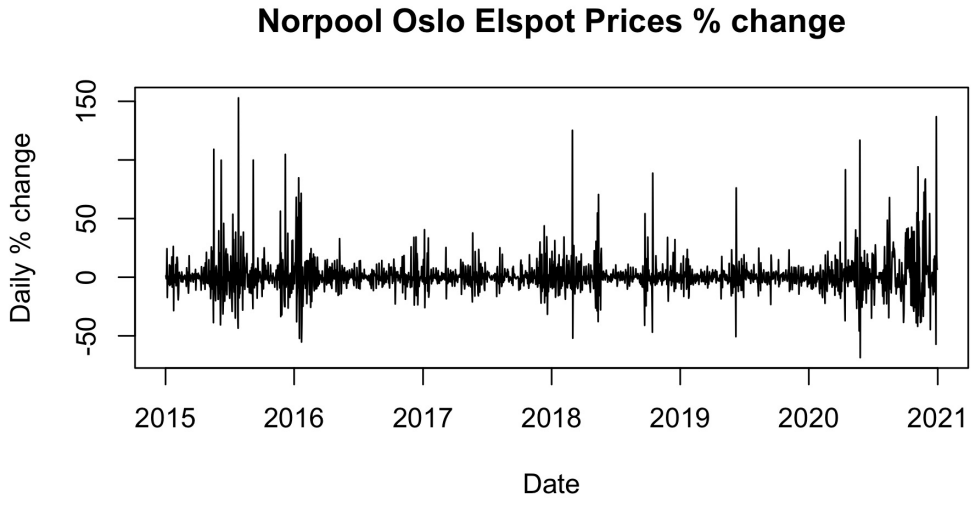
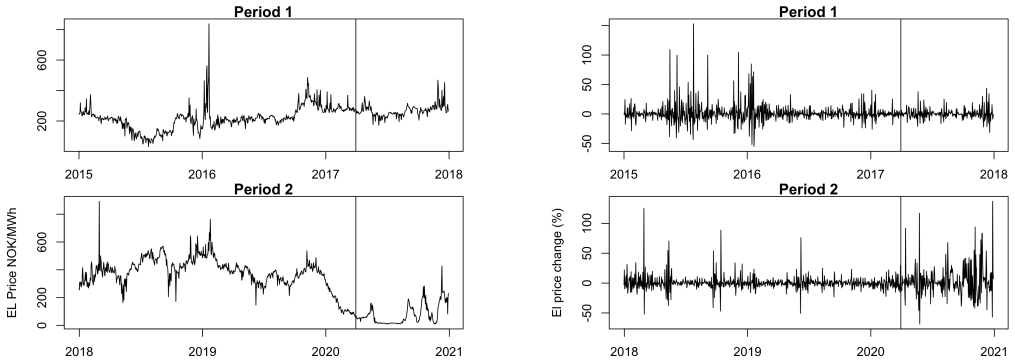


Figure 20: Oslo daily elspot percentage price change from 2015 to 2020

The first period has a couple of periods with very high volatility in 2015

and 2016. The first comes from low summer prices, while the second in the beginning of 2016 comes from winter price spikes. This also coincides with the bottoming of oil prices after a large fall at the same time. For the second period we see a few large volatility spikes in the training data. In the test data prices are mostly very low, and volatility is much more persistent.



(a) EL price level (b) EL price returns

Figure 21: Elspot daily prices and returns for the two subperiods

	Period 1	Period 2
N	1060	1060
Shapiro-Wilk test	$< 2.2e\text{-}16$	$< 2.2e\text{-}16$
Box-Ljung test lag 1	0.005263	$< 2.2e\text{-}16$
Box-Ljung test lag 7	$< 2.7e\text{-}06$	$< 2.2e\text{-}16$
LM test lag 10	< 0.000112	$< 4.466e\text{-}14$

Table 11: Preliminaries

Again, all the tests have very significant p-values for both periods. The coefficient of γ is significant and positive, suggesting volatility increase with falling prices. Volatility is here measured by percentage change, so lower prices also means that a smaller absolute change is necessary for a large percentage change. The number of support vectors is again generally fewer for the wavelet kernel method. There is a strong skewness in the return series, measured at 2.79. The excess kurtosis is measured at a massive 21.17, indicating very

heavy tails. There are significant autocorrelations in the return series for every seven days, for at least one month. A probable explanation is increased demand, and thus higher prices, during weekdays, and lower demand and prices during weekends.

	Model	Estimate	Std. Error	t-value	p-value
Period 1	GJR	0.22981	0.04588	5.009	5.48e-07
	TARCH	0.24641	0.05384	4.577	4.72e-06
Period 2	GJR	0.46340	0.06541	7.085	1.39e-12
	TARCH	0.54786	0.07604	7.205	5.8e-13

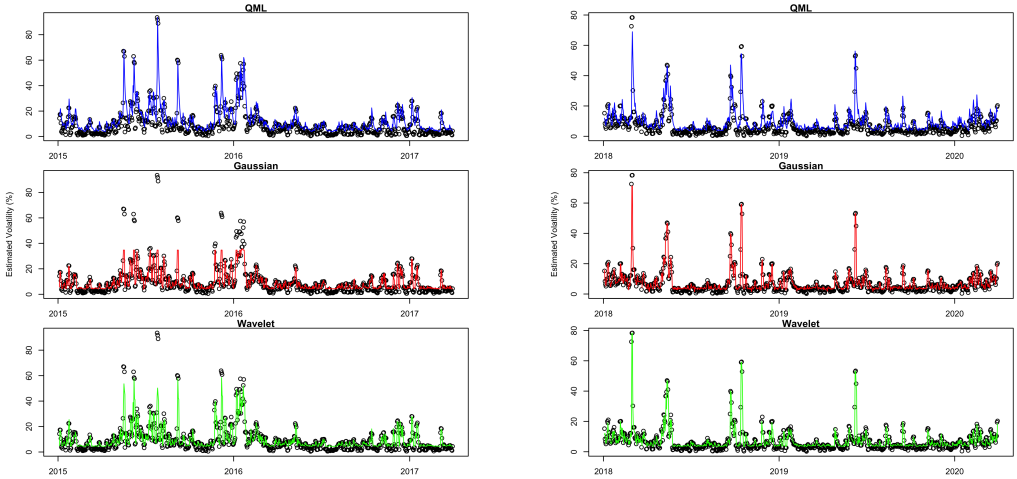
Table 12: Estimate of γ

	Kernel	GARCH	GJR	TARCH	TSGARCH
Period 1	Gaussian	358	472	761	606
	Wavelet	299	437	775	609
Period 2	Gaussian	425	452	627	525
	Wavelet	382	382	625	535

Table 13: Number of support vectors

	Period 1			Period 2		
MSE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	334636.4	308156.2	284960.4	127120.36	27007.23	47353.86
GJR	196976.8	324026.3	259414.9	75178.14	30526.06	45428.81
TARCH	247816.9	338732.9	309233.7	119583.40	48051.89	45511.21
TSGARCH	225003.6	344115.4	320359.2	95928.81	46106.99	44651.23
MAE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	187.9296	117.6003	134.9078	110.76429	22.64757	43.54872
GJR	154.7117	107.7508	125.8024	87.14748	22.36975	41.55503
TARCH	165.2228	120.2940	121.4180	93.99463	32.62379	39.83399
TSGARCH	157.0818	125.6311	127.3699	96.77380	34.15172	41.94148

Table 14: Training Error (MSE & MAE)



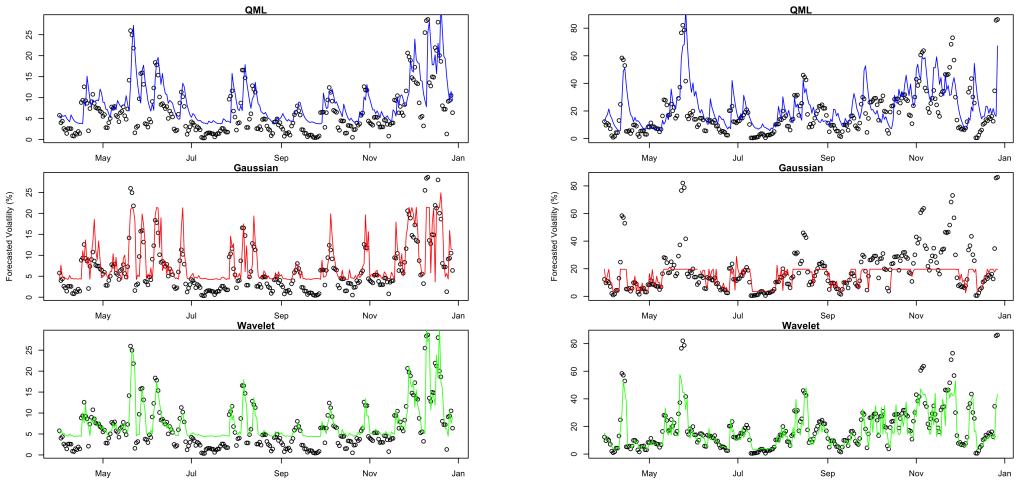
(a) Period 1 (b) Period 2

Figure 22: EL-price daily conditional standard deviation percentage change, approximation and GJR model forecast (training data)

The results of the MSE and MAE are also consistent with the other datasets. The Gaussian kernel method performs best on the training data, while the Wavelet kernel method performs best on the test data. Both SVR methods generally outperform the QML models, in both training and test data. Among the APARCH models the GARCH model seems to perform generally worst, but there is no model consistently outperforming for the kernel methods. For the QML methods the asymmetric GJR and TARCH models have the best test results. Looking at the plots y-axis, there is a large difference in volatility in the two test periods. It seems like the QML models are more reactive to spikes in this dataset compared to the previous two and seems to overreact somewhat to high volatility. They also struggle to capture low volatility period, especially in the second test period. For the second period it looks like the Gaussian model is having similar issues as in the third period of the previous datasets. Instead of a "floor" volatility with upwards spikes there is a volatility "roof" with downwards spikes. The Wavelet kernel looks to perform well on both test data sets.

	Period 1			Period 2		
MSE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	20053.16	10885.670	7582.235	1408664	1293745	790801.4
GJR	16569.51	11371.564	8062.341	1064748	1329058	716285.7
TARCH	15447.13	7877.144	6837.183	1077270	1246207	846736.9
TSGARCH	16545.97	8687.834	6459.655	1254373	1200365	851782.5
MAE	QML	Gaussian	Wavelet	QML	Gaussian	Wavelet
GARCH	78.59977	61.34763	47.49996	589.5322	462.2583	375.0836
GJR	68.64692	59.37889	39.57070	532.4401	464.3968	345.9578
TARCH	70.37944	36.30303	34.41546	545.2083	425.6834	355.5089
TSGARCH	76.39041	38.42772	34.10755	564.7026	407.7899	376.2891

Table 15: Test Error (MSE & MAE)



(a) Period 1 (b) Period 2

Figure 23: EL-price daily conditional standard deviation percentage change, approximation and GJR model forecast (test data)

6 Summary

This thesis has presented the APARCH framework and methods for estimating the models, namely QML and SVR estimation. The models are applied to estimation and forecasting of volatility in financial data to evaluate their empirical performance. The three datasets are the S&P 500 index, High Yield credit spreads and Nordic elspot prices, chosen for their somewhat distinct behavior. For the SVR methods we use the common Gaussian kernel and the wavelet kernel and show that the wavelet kernel can improve the efficiency and forecasting accuracy of the SVR method compared to the Gaussian kernel. Both kernels generally improve both estimation and forecasting performance compared to the QML estimation, although the Gaussian kernel has a problem of overfitting to the datasets studied here.

All periods of all datasets in this study contains asymmetry. The asymmetry coefficient γ from the QML is significant for both the GJR and TARCH models in all periods for all three datasets. For the QML method there is a clear tendency for the GJR and TARCH models, which contain asymmetry effects, to perform better on test data. The SVR method with gaussian kernel is not consistently better than the QML method on test data. The SVR method with wavelet kernel performs consistently better than the QML method in test data for all of the APARCH models, except for the structure breaks in the 2020 data. It is therefore important to include data with different volatility magnitudes when training the model. For the SVR methods there is no evidence of the GJR and TARCH models to perform better on the test data than the symmetric TSGARCH nad GARCH models. The SVR models are quicker to react to increased volatility, and also settles faster after volatility resides. We conclude that the choice of kernel is important to the performance of the SVR method. The wavelet kernel is more adaptable to local clusters and require fewer support vectors is general. Performance on training data is also an important measure. Since the Gaussian kernel

performs best on the traing data, but not better on the test data we have evidence of overfitting. We have a relatively large set of training data in the datasets studied here. It is expected that the wavelet kernel performance would be relatively better on fewer training datapoints, as it generally uses fewer support vectors than the gaussian kernel and the QML models need many observations to estimate the model parameters well.

As we compare the performance of the wavelet and Gaussian kernels to the linear and polynomial kernels (see Appendix) it is clear that there is a problem of overfitting. The wavelet and gaussian kernels have better training errors, but consistently worse test errors. It is therefore evident that the wavelet and Gaussian kernels are too flexible for the data examined here. The linear and polynomial kernels are also capable of capturing the extreme volatility of early 2020 with improved performance to the QML method. A possible alternative is to apply a smoothness prior into the relavance vector machine (Tipping 2001) (RVM), as done in Schmolck and Everson (2007) to enforce sparsity.

References

Aggarwal, S. K., Saini, L. M., & Kumar, A. (2009). Electricity price forecasting in deregulated markets: A review and evaluation. International Journal of Electrical Power & Energy Systems, 31(1), 13-22.

Bezerra, P. C. S., & Albuquerque, P. H. M. (2017). Volatility forecasting via SVR–GARCH with mixture of Gaussian kernels. Computational Management Science, 14(2), 179-196.

Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. Journal of political economy, 81(3), 637-654.

Bollerslev, T., (1986), Generalized autoregressive conditional heteroskedasticity, Journal of Econometrics 31, 307-327.

Bollerslev, T. (1987). A conditionally heteroskedastic time series model for speculative prices and rates of return. The review of economics and statistics, 542-547.

Bollerslev, T. and Wooldridge, J. M. (1992) Quasi-maximum likelihood estimation and inference in dynamic models with time-varying covariances. Econometric Reviews 11: 143–172.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992, July). A training algorithm for optimal margin classifiers. In Proceedings of the fifth annual workshop on Computational learning theory (pp. 144-152).

Chen, S., Härdle, W. K., & Jeong, K. (2010). Forecasting volatility with support vector machine-based GARCH model. Journal of Forecasting, 29(4), 406-433.

Cifter, A. (2013). Forecasting electricity price volatility with the Markov-switching GARCH model: Evidence from the Nordic electric power market. Electric Power Systems Research, 102, 61-67.

Clark, E., & Baccar, S. (2018). Modelling credit spreads with time volatility, skewness, and kurtosis. Annals of Operations Research, 262(2), 431-461.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.

Daubechies, I. (1988). Orthonormal bases of compactly supported wavelets. Communications on pure and applied mathematics, 41(7), 909-996.

Ding, Z., Granger, C. W. J., and Engle, R. (1993) A long memory property of stock market returns and a new model. Journal of Empirical Finance 1 83-106.

dos Santos Coelho, L., & Santos, A. A. (2011). A RBF neural network model with GARCH errors: application to electricity price forecasting. Electric Power Systems Research, 81(1), 74-83.

Drucker, H., Burges, C. J., Kaufman, L., Smola, A., & Vapnik, V. (1996). Support vector regression machines. Advances in neural information processing systems, 9, 155-161.

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. Econometrica: Journal of the Econometric Society, 987-1007.

Engle, R. F. and González-Rivera, G. (1991) Semi parametric ARCH models. Journal of Business and Economic Statistics 9: 345–359.

Glosten, L. R., Jagannathan, R. and Runkle, D. E. (1993) On the relation between expected value and the volatility of the nominal excess return on stocks Journal of Finance 48 No. 5: 1779–1801.

Grossmann, A., & Morlet, J. (1984). Decomposition of Hardy functions into square integrable wavelets of constant shape. SIAM journal on mathematical analysis, 15(4), 723-736.

Haar, A. (1909). Zur theorie der orthogonalen funktionensysteme. Georg-August-Universitat, Gottingen.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). An Introduction to Statistical Learning. Springer.

Karush, W. (1939). Minima of functions of several variables with inequalities as side constraints. M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago.

Kuhn, H. W., Tucker, A. W. (1951).Nonlinear Programming. Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 481-492,

Li, Y. S. (2014) Estimating APGARCH-Skew-t model by Wavelet Support Vector Machines. Journal of Forecasting 33(4): 259–269.

Li, Y. S., Karlsson, H. K. (2020) Estimating volatility in oil price by APARCH - SVR models. Working Paper

Lintner, J. (1965). Security prices, risk, and maximal gains from diversification. The journal of finance, 20(4), 587-615.

Mallat, S. G. (1989). A theory for multiresolution signal decomposition: the wavelet representation. IEEE transactions on pattern analysis and machine intelligence, 11(7), 674-693.

Markowitz, H. (1952). The utility of wealth. Journal of political Economy, 60(2), 151-158.

Mercer, J. (1909) Functions of positive and negative type and their connection with the theory of integral equations. Philosophical Transactions of the Royal Society, London, A 209: 415–446.

Mossin, J. (1966). Equilibrium in a capital asset market. Econometrica: Journal of the econometric society, 768-783.

Nelson, D. B. (1991). Conditional heteroskedasticity in asset returns: A new approach. Econometrica: Journal of the Econometric Society, 347-370.

Olowe, R. A. (2009). Modelling naira/dollar exchange rate volatility: Application of GARCH and asymmetric models. International Review of Business Research Papers, 5(3), 377-398.

Ou, P., & Wang, H. (2010). Financial volatility forecasting by least square support vector machine based on GARCH, EGARCH and GJR models: evidence from ASEAN stock markets. International Journal of Economics and Finance, 2(1), 51-64.

Peng, Y., Albuquerque, P. H. M., de Sá, J. M. C., Padula, A. J. A., & Montenegro, M. R. (2018). The best of two worlds: Forecasting high frequency volatility for cryptocurrencies and traditional currencies with Support Vector Regression. Expert Systems with Applications, 97, 177-192.

Pérez-Cruz, F., Afonso-Rodriguez, J. A., & Giner, J. (2003). Estimating garch models using support vector machines*. Quantitative Finance, 3(3), 163-172.

Sayad, Dr. S. (n.d.). Support Vector Machine - Regression (SVR). Retrieved January 14, 2021, from https://www.saedsayad.com/support_vector_machine_reg.htm

Schmolck, A., & Everson, R. (2007). Smooth relevance vector machine: a smoothness prior extension of the RVM. Machine Learning, 68(2), 107-135.

Schwert, G. W. (1989). Why does stock market volatility change over time?. The journal of finance, 44(5), 1115-1153.

Schölkopf, B., Smola, A. J., & Bach, F. (2002). Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector

regression. Statistics and computing, 14(3), 199-222.

Shapiro, S. S. and Wilk, M. B. (1965). An Analysis of Variance Test for Normality (Complete Samples). Biometrika 52: 591–611.

Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. The journal of finance, 19(3), 425-442.

Stavroyiannis, S. (2016). Value-at-Risk and backtesting with the APARCH model and the standardized Pearson type IV distribution. Available at SSRN 2734058.

Sun, H., & Yu, B. (2020). Forecasting Financial Returns Volatility: A GARCH-SVR Model. Computational Economics, 55(2), 451-471.

Suykens, J. A., & Vandewalle, J. (1999). Least squares support vector machine classifiers. Neural processing letters, 9(3), 293-300.

Tang, L. B., Tang, L. X., & Sheng, H. Y. (2009). Forecasting volatility based on wavelet support vector machine. Expert Systems with Applications, 36(2), 2901-2909.

Taylor, S. (2000). (1986), Modelling Financial Time Series.

Tipping, M. E. (2001). Sparse Bayesian learning and the relevance vector machine. Journal of machine learning research, 1(Jun), 211-244.

Treynor, J. L. (1961). Market value, time, and risk. Time, and Risk (August 8, 1961).

Treynor, J.L. (1962) Toward a Theory of Market Value of Risk Assets (Unpublished Manuscript, a Revised Version Was Published in Korajczyk, Robert, A, Ed., 1999, Asset Pricing and Portfolio Performance: Models, Strategy and Performance Metrics, Risk Books, London, 15-22).

Vapnik, V., Lerner, A. (1963). Pattern recognition using generalized portrait method. Automation and remote control, 24, 774-780.

Vapnik, V. and Chervonenkis A. (1964). A note on one class of perceptrons. Automation and Remote Control, 25.

Vapnik, V. and Chervonenkis, A. (1974) Theory of Pattern Recognition (in Russian). Nauka: Moscow

Vapnik, V. 1982. Estimation of Dependences Based on Empirical Data. Springer: Berlin.

Vapnik, V. (1995) The Nature of Statistical Learning Theory. Springer: New York.

Vapnik, V. (1992). Principles of risk minimization for learning theory. In Advances in neural information processing systems (pp. 831-838).

Ben-Hur, A., Horn, D., Siegelmann, H. T., & Vapnik, V. (2001). Support vector clustering. Journal of machine learning research, 2(Dec), 125-137.

Zhang, L., Zhou, W. D. and Jiao, L. C. (2004) Wavelet support vector machine. IEEE Transactions on Systems, Man, and Cybernetics Part B 34(1): 34–39.

Appendix

S&P 500

Period 1	MSE				
	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	16.81715	1.32472	8.71106	14.64294	14.64349
GJR	15.69177	1.22932	8.14722	14.61479	14.61386
TARCH	15.40920	4.32664	10.55776	14.74244	14.74241
TSGARCH	18.87404	3.12808	9.79917	15.11812	15.11907
	MAE				
	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	1.66402	0.37845	0.99268	1.26842	1.26835
GJR	1.59100	0.35872	0.91649	1.26695	1.26695
TARCH	1.63374	0.63234	1.00173	1.26083	1.26083
TSGARCH	1.73324	0.67384	1.08036	1.28442	1.28438

Table 16: S&P 500 period 1 training error

	Period 1				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	1.51006	3.77668	1.07864	1.04818	1.04831
GJR	1.39698	4.20442	1.14557	1.04588	1.04569
TARCH	1.48663	1.65659	1.19918	1.07499	1.07499
TSGARCH	1.52570	1.45023	1.10210	1.06198	1.06208
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	0.88765	0.98843	0.58218	0.58118	0.58113
GJR	0.85132	1.05469	0.59407	0.58227	0.58227
TARCH	0.86606	0.66888	0.58669	0.57474	0.57474
TSGARCH	0.89269	0.64599	0.57954	0.58472	0.58467

Table 17: S&P 500 period 1 test Error

	Period 2				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	1.64263	0.80463	1.01889	1.03453	1.03437
GJR	1.25389	0.30887	0.85007	1.04224	1.04215
TARCH	1.64690	0.74614	0.97648	1.04993	1.04971
TSGARCH	2.08243	0.68665	0.99345	1.04493	1.04495
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	0.51935	0.26937	0.37978	0.39143	0.39143
GJR	0.49654	0.21399	0.34225	0.38831	0.38830
TARCH	0.52440	0.30809	0.35291	0.38881	0.38881
TSGARCH	0.55076	0.31391	0.37471	0.39215	0.39214

Table 18: S&P 500 period 2 training error

	Period 2				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	1.09265	0.87601	0.55606	0.44731	0.44728
GJR	0.79828	0.94426	0.42858	0.43414	0.43416
TARCH	0.92261	0.56889	0.49906	0.42541	0.42552
TSGARCH	1.25320	0.75262	0.58780	0.44612	0.44614
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	0.57520	0.44372	0.38258	0.35709	0.35707
GJR	0.49856	0.47754	0.36921	0.35603	0.35601
TARCH	0.52932	0.40217	0.38072	0.34949	0.34950
TSGARCH	0.61574	0.42473	0.38952	0.35314	0.35314

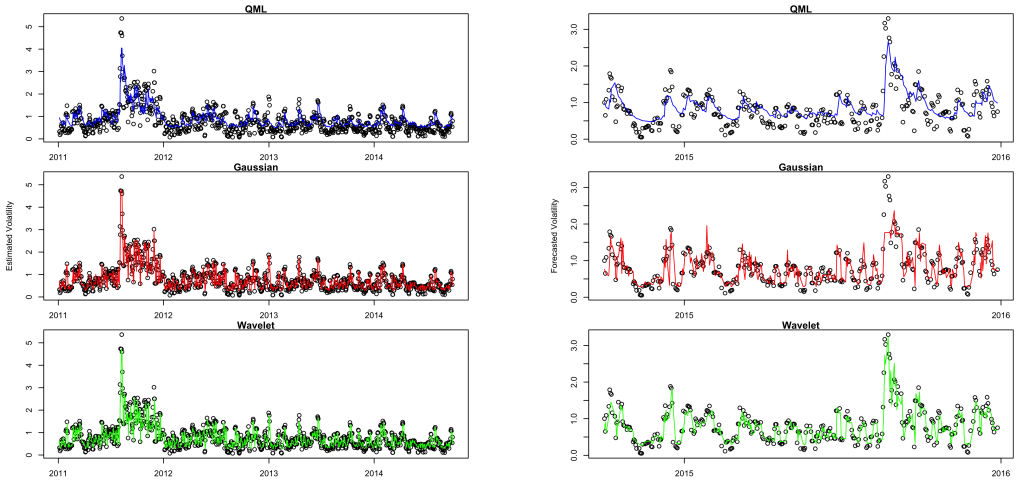
Table 19: S&P 500 period 2 test error

	Period 3				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	0.53044	0.08023	0.31761	0.43835	0.43822
GJR	0.49812	0.06708	0.24358	0.42360	0.42565
TARCH	0.55091	0.17364	0.34106	0.43237	0.43197
TSGARCH	0.65626	0.16714	0.38384	0.44315	0.44311
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	0.36890	0.09394	0.23745	0.28214	0.28213
GJR	0.37197	0.07673	0.21045	0.27922	0.28026
TARCH	0.39436	0.15159	0.23384	0.28143	0.28141
TSGARCH	0.40641	0.16174	0.25334	0.28381	0.28380

Table 20: S&P 500 period 3 training error

	Period 3				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	51.70217	124.5159	113.6029	23.31097	23.30867
GJR	48.51570	125.3363	111.9718	24.03008	23.92146
TARCH	50.11123	124.4605	118.5207	24.25303	24.24577
TSGARCH	57.14117	122.1579	108.0572	24.13403	24.11614
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	2.25864	3.39925	2.89213	1.49143	1.49159
GJR	2.11129	3.35681	2.93830	1.49905	1.50318
TARCH	2.15640	3.26802	3.02260	1.48911	1.48950
TSGARCH	2.34104	3.16420	2.71566	1.48932	1.48926

Table 21: S&P 500 period 3 test error



(a) Training data (b) Test data

Figure 24: S&P 500 daily conditional standard deviation change in percent, approximation and GJR model estimate/forecast, period 2

Credit Spreads

	Period 1				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	218496.5	4360.792	101785.6	159846.5	159847.6
GJR	239002.6	3730.750	100955.9	158892.5	158953.7
TARCH	255528.7	27578.188	106003.5	168723.0	168743.9
TSGARCH	243880.8	25766.276	112137.5	168884.7	168917.6
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	163.3819	35.59899	88.82319	110.1894	110.2009
GJR	169.8250	22.51848	78.64740	106.0367	105.9592
TARCH	174.2990	44.60232	77.13425	106.7046	106.7047
TSGARCH	169.6626	45.89906	84.44192	106.7267	106.7280

Table 22: Credit spreads period 1 training error

	Period 1				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	6887.091	18167.89	2886.892	2849.297	2849.957
GJR	7410.604	30890.71	3062.950	2754.615	2753.602
TARCH	8352.216	25266.90	3165.091	2859.813	2858.088
TSGARCH	7732.864	13743.41	3303.745	2867.481	2865.775
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	51.02618	66.61903	33.59555	33.91555	33.92864
GJR	53.39764	83.19432	29.41796	29.52870	29.44881
TARCH	56.86333	44.92550	28.96660	28.16569	28.15724
TSGARCH	54.95110	39.40501	29.11947	28.17686	28.17054

Table 23: Credit spreads preiod 1 test error

	Period 2				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	8101.009	810.0459	3674.282	4603.731	4603.726
GJR	7748.410	2697.7992	4269.155	4588.186	4588.005
TARCH	8402.395	1509.7969	3930.245	4716.942	4717.074
TSGARCH	8972.592	1362.7958	4007.754	4759.633	4759.615
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	38.19850	14.90734	22.73174	25.17883	25.17861
GJR	38.76249	17.37126	23.89126	25.24115	25.24678
TARCH	38.87981	16.79760	21.62147	24.76067	24.75656
TSGARCH	38.29345	16.36505	22.58444	24.77432	24.77441

Table 24: Credit spreads period 2 training error

	Period 2				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	11293.390	8083.478	6631.966	4768.987	4768.995
GJR	9879.722	8518.108	5004.100	4727.968	4726.650
TARCH	9948.205	6558.134	6148.325	4810.760	4810.746
TSGARCH	11504.047	7320.353	5338.058	4887.137	4887.181
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	63.01395	45.24267	39.15554	34.73566	34.73564
GJR	57.96255	46.13750	36.28041	34.48453	34.47840
TARCH	58.17731	40.54319	38.10351	34.28531	34.28341
TSGARCH	63.22833	41.58775	36.08725	34.68558	34.68589

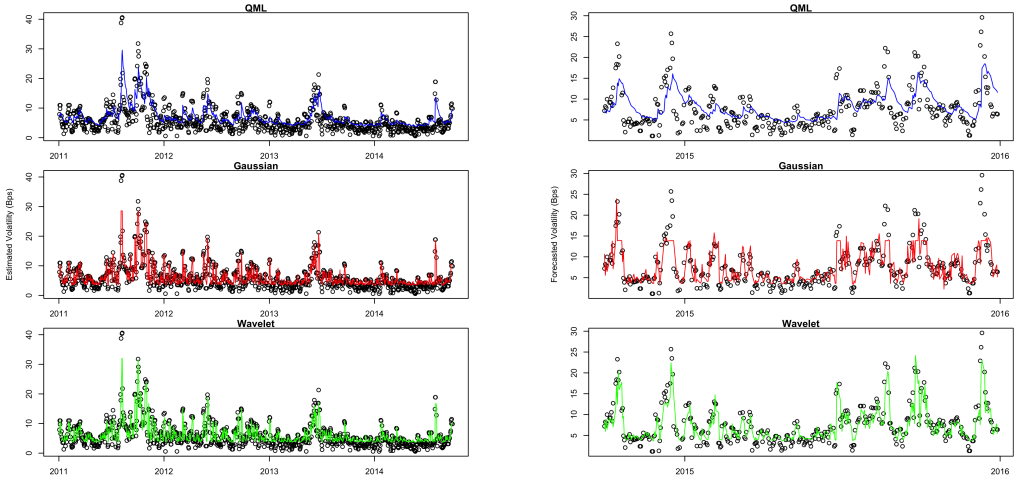
Table 25: Credit spreads period 2 test error

	Period 3				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	4769.588	1910.875	2825.166	3643.113	3643.028
GJR	5056.957	1303.051	2703.521	3648.215	3648.073
TARCH	5189.049	1876.403	2315.692	2795.324	2795.217
TSGARCH	4687.670	1913.564	2497.677	2820.901	2820.624
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	34.41038	12.66757	20.63566	24.42307	24.42309
GJR	35.76593	11.31636	19.70586	24.41339	24.41350
TARCH	35.23777	15.17021	18.89051	21.91041	21.91017
TSGARCH	33.66951	15.77763	20.05955	21.89380	21.89378

Table 26: Credit spreads period 3 training rror

	Period 3				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	273732.0	777721.1	710181.6	204142.3	204185.0
GJR	267752.7	784100.4	716217.9	202983.3	202983.5
TARCH	325126.4	769984.0	760646.6	224139.6	224237.3
TSGARCH	372827.9	770903.7	727815.7	217543.9	217537.0
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	202.8435	296.5185	270.4561	144.7150	144.7279
GJR	207.4479	302.0602	269.3533	144.2403	144.2370
TARCH	227.2722	285.6801	281.3547	150.4788	150.5170
TSGARCH	226.9181	287.1965	272.8187	146.9875	146.9880

Table 27: Credit spreads period 3 test error



(a) Training data (b) Test data

Figure 25: HY spred daily conditional standard deviation bps change, approximation and GJR model estimate/forecast, period 2

EL Prices

	Period 1				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	334636.4	308156.2	284960.4	241816.9	241817.4
GJR	196976.8	324026.3	259414.9	237378.9	237355.9
TARCH	247816.9	338732.9	309233.7	250534.4	250568.8
TSGARCH	225003.6	344115.4	320359.2	252420.7	252425.7
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	187.9296	117.6003	134.9078	130.7509	130.7488
GJR	154.7117	107.7508	125.8024	122.6200	122.6339
TARCH	165.2228	120.2940	121.4180	123.7075	123.7055
TSGARCH	157.0818	125.6311	127.3699	125.5982	125.6016

Table 28: EL-Price period 1 training error

	Period 1				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	20053.16	10885.670	7582.235	6674.261	6674.169
GJR	16569.51	11371.564	8062.341	6661.196	6660.605
TARCH	15447.13	7877.144	6837.183	6680.229	6681.889
TSGARCH	16545.97	8687.834	6459.655	6487.855	6488.221
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	78.59977	61.34763	47.49996	45.60304	45.60020
GJR	68.64692	59.37889	39.57070	36.75635	36.77554
TARCH	70.37944	36.30303	34.41546	33.50307	33.50317
TSGARCH	76.39041	38.42772	34.10755	33.71357	33.71122

Table 29: EL-Price period 1 test error

	Period 2				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	127120.36	27007.23	47353.86	75312.39	75312.18
GJR	75178.14	30526.06	45428.81	56969.84	56974.58
TARCH	119583.40	48051.89	45511.21	66600.54	66620.02
TSGARCH	95928.81	46106.99	44651.23	70846.30	70864.51
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	110.76429	22.64757	43.54872	57.13660	57.13749
GJR	87.14748	22.36975	41.55503	52.20520	52.21122
TARCH	93.99463	32.62379	39.83399	53.74743	53.76504
TSGARCH	96.77380	34.15172	41.94148	55.58962	55.58922

Table 30: EL-Price period 2 training error

	Period 2				
MSE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	1408664	1293745	790801.4	417442.5	417441.9
GJR	1064748	1329058	716285.7	304335.5	304346.8
TARCH	1077270	1246207	846736.9	373924.0	374127.4
TSGARCH	1254373	1200365	851782.5	405586.1	405594.2
MAE	QML	Gaussian	Wavelet	Linear	Polynomial
GARCH	589.5322	462.2583	375.0836	258.7513	258.7511
GJR	532.4401	464.3968	345.9578	259.6291	259.6465
TARCH	545.2083	425.6834	355.5089	246.6530	246.7129
TSGARCH	564.7026	407.7899	376.2891	256.3015	256.2866

Table 31: EL-Price period 2 test error

R-code

```
##R-code for the S\&P 500 dataset. For the other datasets the code is the same,
#except for the data loading and preparation.
require(readxl)
require(moments)
require(ggplot2)
library(FinTS)
library(lmtest)
library(TTR)
require(e1071)
require(kernlab)
require("fGarch")
require(quantmod)

#load data
getSymbols(Symbols = "^GSPC", src = "yahoo", from = "2006-01-01", to = "2020-12-31", periodicity

#Candlestick chart of the S&P 500 index from 2005 to today in linear and log scale, and
#for the last year
barChart(GSPC)
png("/Users/Arne/Documents/STAT/STAT200/images/SP500.png", units = "cm", width = 16, height = 9,
barChart(GSPC,log.scale = TRUE, TA = NULL, theme = "white")
dev.off()
n = length(GSPC)/6
barChart(GSPC[-(1:(n-250))])

#The daily return series in %
y=(exp(diff(log(GSPC$GSPC.Close)))-1)
y=y[-1,]*100
png("/Users/Arne/Documents/STAT/STAT200/images/SPchg.png", units = "cm", width = 16, height = 9,
plot(y,xlab="Date",ylab="Returns",axes=FALSE, main = "S&P 500 daily % change", lwd = .8, log = F
dev.off()

#The Wavelet kernel, Zhang et al. (2004)
k = function(x,y) {prod(cos(1.75*(x-y)/2)*exp(-(x-y)^2/8))}
class(k) = "kernel"

u = y
u = as.ts(u$GSPC.Close)
shapiro.test(u) #Normality test, rejected
ArchTest(u,lag=12) #LM test, rejected
skewness(u)
```

```
kurtosis(u)

Box.test(u, lag = 1, type = "Ljung") #Box Ljung test
Box.test(u^2, lag = 1, type = "Ljung")
Box.test(u^2, lag = 5, type = "Ljung")
acf(u)
acf(u^2)

acff = acf(abs(u), lag.max = 500) #autocorrelation function
acff = acf(u^2, lag.max = 500)

#Divide dataset into three separate periods.
u1=u[1:(length(u)/3)] #u: return series.
u2=u[(length(u)/3+1):(2*length(u)/3)]
u3=u[(2*length(u)/3+1):length(u)]
x1=index(GSPC[1:(length(u)/3)]) #x: date series.
x2=index(GSPC[(length(u)/3+1):(2*length(u)/3)])
x3=index(GSPC[(2*length(u)/3+1):length(u)])
p1=GSPC$GSPC.Close[1:(n/3)] #p: price series.
p2=GSPC$GSPC.Close[(n/3+1):(2*n/3)]
p3=GSPC$GSPC.Close[(2*n/3+1):n]

png("/Users/Arne/Documents/STAT/STAT200/images/SPprices.png", units = "cm", width = 16, height =
par(mfrow=c(3,1), mar=c(2,4,1,1))
plot(x1,p1, xlab="", ylab = "", main = "Period 1", type = "l")
abline(v=x1[length(x1)*3/4])

plot(x2,p2, xlab = "", ylab = " S&P 500 Price", main = "Period 2", type = "l")
abline(v=x2[length(x1)*3/4])

plot(x3,p3, xlab="", ylab = "", main = "Period 3", type = "l")
abline(v=x3[length(x1)*3/4])
dev.off()

png("/Users/Arne/Documents/STAT/STAT200/images/SPreturns.png", units = "cm", width = 16, height
par(mfrow=c(3,1), mar=c(2,4,1,1))
plot(x1,u1, xlab="", ylab = "", main = "Period 1", type = "l")
abline(v=x1[length(x1)*3/4])

plot(x2,u2, xlab = "", ylab = "S&P 500 Returns (%)", main = "Period 2", type = "l")
abline(v=x2[length(x1)*3/4])

plot(x3,u3, xlab="", ylab = "", main = "Period 3", type = "l")
abline(v=x3[length(x1)*3/4])
dev.off()
```

```
#Repeat tests for all time periods
shapiro.test(u1)
shapiro.test(u2)
shapiro.test(u3)

Box.test(u, lag = 1000, type = "Ljung")
Box.test(u1^2, lag = 1, type = "Ljung")
Box.test(u2^2, lag = 1, type = "Ljung")
Box.test(u3^2, lag = 1, type = "Ljung")

Box.test(u1^2, lag = 5, type = "Ljung")
Box.test(u2^2, lag = 5, type = "Ljung")
Box.test(u3^2, lag = 5, type = "Ljung")


ArchTest(u1,lag=10)
ArchTest(u2,lag=10)
ArchTest(u3,lag=10)

#decide which timeperiod to investigate
u = u2
x = x2

plot(x = x, u, type = "h", main = "S&P 500 daily % change", ylab = "%", xlab = "Date")
(T=length(u))
hhat = SMA(u^2,n = 3) #estimate of "true" h
plot(x = x, u^2, type = "l", xlab = "")
lines(x = x, hhat, type = "l", col = 3)
plot(x = x, hhat-u^2, type = "l")

M = 1*T/4


#GARCH

#Divide training and test dataset
utrain=u[4:(T-M)]-mean(u[4:(T-M)])
utest=u[((T-M)+1):(T-1)]-mean(u[((T-M)+1):(T-1)])
xTr = x[4:(T-M)]
xTs = x[((T-M)+1):(T-1)]

xtrain=cbind(u[3:((T-M)-1)]^2,hhat[3:((T-M)-1)]) #input data for SVR, training data
hhattrain=hhat[4:(T-M)]
```

```
plot(xTr,utrain^2,type="l", col="red")
lines(xTr,hhattrain, type = "h")


xtest=cbind(u[((T-M)+1):(T-1)]^2,hhat[((T-M)+1):(T-1)]) #input data for SVR, test data
hhattest=hhat[((T-M)+2):(T)]

plot(xTs,utest^2,type="l", col="red")
lines(xTs,hhattest, type = "l")

#Find the best values of C and epsilon for the SVR, using the training data, by grid search.
tobj=tune.svm(x=xtrain,y=hhattrain,cost=c(2,5,25,50),epsilon=c(0.005,0.01,.025,.1,0.25))
tobj$best.parameters

Cvalue=tobj$best.parameters$cost
epsvalue=tobj$best.parameters$epsilon

#Fit GARCH-SVR with Gaussian, wavelet, linear and polynomial kernel respectively using 5-fold
#validation
regm <-ksvm(xtrain,hhattrain,epsilon=epsvalue,C=Cvalue,cross=5)
regmnew <-ksvm(xtrain,hhattrain,epsilon=epsvalue,kernel=k,C=Cvalue,cross=5)
regmlin = ksvm(xtrain,hhattrain,epsilon=epsvalue,C=Cvalue,kernel="vanilladot",cross=5)
regmpoly = ksvm(xtrain,hhattrain,epsilon=epsvalue,C=Cvalue,kernel="polydot",cross=5)

#MLE GARCH
fit2=garchFit(~ garch(1,1), data =utrain, delta=2, include.mean=FALSE, trace = FALSE)
fit2
volatility=slot(fit2, "h.t")

plot(xTr,hhattrain,type="p")
lines(xTr,volatility,col="blue")
plot(xTr,hhattrain,type="p")
lines(xTr,predict(regmnew,xtrain)[1:LTr],col="blue")
plot(xTr,hhattrain,type="p")
lines(xTr,predict(regm,xtrain)[1:LTr],col="blue")

LTr=length(hhattrain)
#MSE and MAE calculation
garchtrainmse=c(sum((volatility[1:LTr]-hhattrain[1:LTr])^2)/(LTr-(1-1)),
                sum((predict(regm,xtrain)[1:LTr]-hhattrain[1:LTr])^2)/(LTr-(1-1)),
                sum((predict(regmnew,xtrain)[1:LTr]-hhattrain[1:LTr])^2)/(LTr-(1-1)),
                sum((predict(regmlin,xtrain)[1:LTr]-hhattrain[1:LTr])^2)/(LTr-(1-1)),
                sum((predict(regmpoly,xtrain)[1:LTr]-hhattrain[1:LTr])^2)/(LTr-(1-1)))
```

```
garchtrainmse

garchtrainabs=c(sum(abs(volatility[1:LTr]-hhattrain[1:LTr]))/(LTr-(1-1)),
                sum(abs(predict(regm,xtrain)[1:LTr]-hhattrain[1:LTr]))/(LTr-(1-1)),
                sum(abs(predict(regmnew,xtrain)[1:LTr]-hhattrain[1:LTr]))/(LTr-(1-1)),
                sum(abs(predict(regmlin,xtrain)[1:LTr]-hhattrain[1:LTr]))/(LTr-(1-1)),
                sum(abs(predict(regmpoly,xtrain)[1:LTr]-hhattrain[1:LTr]))/(LTr-(1-1)))

garchtrainabs

#error plot
plot(abs(volatility[1:LTr]-hhattrain[1:LTr]))
plot(abs(predict(regm,xtrain)[1:LTr]-hhattrain[1:LTr]))
plot(abs(predict(regmnew,xtrain)[1:LTr]-hhattrain[1:LTr]))
summary(abs(predict(regm,xtrain)[1:LTr]-hhattrain[1:LTr]))
summary(abs(predict(regmnew,xtrain)[1:LTr]-hhattrain[1:LTr]))

plot(xTs,hhattest,type="p")
lines(xTs,predict(regm,xtest),col="red")

plot(xTs,hhattest,type="p")
lines(xTs,predict(regmnew,xtest),col="green")

#Calulate the volatility forecast using the MLE GARCH model parameters
B <- coef(fit2)
hgpre=numeric(length(hhattest))
hgpre[1]=hhattest[1]
for (i in 1:(length(hhattest)-1)){
  hgpre[i+1]=B[1]+B[2]*(utest[i]^2)+B[3]*hgpre[i]
}

plot(xTs,hhattest,type="p")
lines(xTs,hgpre,col="blue")

#MSE and MAE calculation
garchtestmse=c(sum((hhattest-hgpre)^2)/length(hhattest),
               sum((predict(regm,xtest)-hhattest)^2)/length(hhattest),
               sum((predict(regmnew,xtest)-hhattest)^2)/length(hhattest),
               sum((predict(regmlin,xtest)-hhattest)^2)/length(hhattest),
               sum((predict(regmpoly,xtest)-hhattest)^2)/length(hhattest))
garchtestmse

garchtestabs=c(sum(abs(hhattest-hgpre))/length(hhattest),
               sum(abs(predict(regm,xtest)-hhattest))/length(hhattest),
               sum(abs(predict(regmnew,xtest)-hhattest))/length(hhattest),
```

```
                sum(abs(predict(regmlin,xtest)-hhattest))/length(hhattest),
                sum(abs(predict(regmpoly,xtest)-hhattest))/length(hhattest))
garchtestabs

plot(abs(predict(regm,xtest)-hhattest))
plot(abs(predict(regmnew,xtest)-hhattest))
plot(abs(hhattest-hgpre))


#GJR
xtrainGJR=cbind(u[3:((T-M)-1)]^2, u[3:((T-M)-1)]*abs(u[3:((T-M)-1)]),hhat[3:((T-M)-1)])
hhattrainGJR=hhat[4:(T-M)]
xtestGJR=cbind(u[((T-M)+1):(T-1)]^2,u[((T-M)+1):(T-1)]*abs(u[((T-M)+1):(T-1)]),hhat[((T-M)+1):(T
hhattestGJR=hhat[((T-M)+2):(T)]


tobjGJR=tune.svm(x=xtrainGJR,y=hhattrainGJR,cost=c(2,5,10,25),epsilon=c(.01,.025,0.05,.1))
tobjGJR$best.parameters

CvalueGJR=tobjGJR$best.parameters$cost
epsvalueGJR=tobjGJR$best.parameters$epsilon

regmGJR <-ksvm(xtrainGJR,hhattrainGJR,epsilon=epsvalueGJR,C=CvalueGJR,cross=5)
regmnewGJR <-ksvm(xtrainGJR,hhattrainGJR,epsilon=epsvalueGJR,kernel=k,C=CvalueGJR,cross=5)
regmlin = ksvm(xtrainGJR,hhattrainGJR,epsilon=epsvalueGJR,C=CvalueGJR,kernel="vanilladot",cross=
regmpoly = ksvm(xtrainGJR,hhattrainGJR,epsilon=epsvalueGJR,C=CvalueGJR,kernel="polydot",cross=5)
#k

fitGJR=garchFit(~ garch(1,1), data = utrain, delta=2, leverage=TRUE,include.mean=FALSE, trace =
fitGJR
volatilityGJR = slot(fitGJR, "h.t")

#plots the estimated volatility against the approximation
png("/Users/Arne/Documents/STAT/STAT200/images/SP2Train.png", units = "cm", width = 16, height =
par(mfrow=c(3,1), mar=c(2,4,1,1))
plot(xTr,sqrt(hhattrainGJR), xlab="", ylab = "", main = "QML")
lines(xTr,sqrt(volatilityGJR), col = "blue")

plot(xTr,sqrt(hhattrainGJR), xlab = "", ylab = "Estimated Volatility", main = "Gaussian")
lines(xTr,sqrt(predict(regmGJR,xtrainGJR)),col="red")

plot(xTr,sqrt(hhattrainGJR), xlab="", ylab = "", main = "Wavelet")
lines(xTr,sqrt(predict(regmnewGJR,xtrainGJR)),col="green")
dev.off()
BI=1
```

```
#MSE and MAE calculation
GJRtrainmse=c(
  sum((volatilityGJR[BI:LTr]-hhattrainGJR[BI:LTr])^2)/(LTr-(BI-1)),
  sum((predict(regmGJR,xtrainGJR)[BI:LTr]-hhattrainGJR[BI:LTr])^2)/(LTr-(BI-1)),
  sum((predict(regmnewGJR,xtrainGJR)[BI:LTr]-hhattrainGJR[BI:LTr])^2)/(LTr-(BI-1))
  sum((predict(regmlin,xtrainGJR)[BI:LTr]-hhattrainGJR[BI:LTr])^2)/(LTr-(BI-1)),
  sum((predict(regmpoly,xtrainGJR)[BI:LTr]-hhattrainGJR[BI:LTr])^2)/(LTr-(BI-1))
  )
GJRtrainmse

GJRtrainabs=c(
  sum(abs(volatilityGJR[BI:LTr]-hhattrainGJR[BI:LTr]))/(LTr-(BI-1)),
  sum(abs(predict(regmGJR,xtrainGJR)[BI:LTr]-hhattrainGJR[BI:LTr]))/(LTr-(BI-1)),
  sum(abs(predict(regmnewGJR,xtrainGJR)[BI:LTr]-hhattrainGJR[BI:LTr]))/(LTr-(BI-1)),
  sum(abs(predict(regmlin,xtrainGJR)[BI:LTr]-hhattrainGJR[BI:LTr]))/(LTr-(BI-1)),
  sum(abs(predict(regmpoly,xtrainGJR)[BI:LTr]-hhattrainGJR[BI:LTr]))/(LTr-(BI-1)))
GJRtrainabs

plot(hhattestGJR,type="p")
lines(predict(regmGJR,xtestGJR),col="red")


BGJR <- coef(fitGJR)
hgpreGJR=numeric(length(hhattestGJR))
hgpreGJR[1]=hhattestGJR[1]
for (i in 1:(length(hhattestGJR)-1)){
  hgpreGJR[i+1]=BGJR[1]+BGJR[2]*(abs(utest[i])-BGJR[3]*utest[i])^2+BGJR[4]*hgpreGJR[i]
}

#plots the forecasted volatility against the approximation
png("/Users/Arne/Documents/STAT/STAT200/images/SP2Test.png", units = "cm", width = 16, height =
par(mfrow=c(3,1), mar=c(2,4,1,1))
plot(xTs,sqrt(hhattestGJR), xlab="", ylab = "", main = "QML")
lines(xTs,sqrt(hgpreGJR), col = "blue")

plot(xTs,sqrt(hhattestGJR), xlab = "", ylab = "Forecasted Volatility", main = "Gaussian")
lines(xTs,sqrt(predict(regmGJR,xtestGJR)),col="red")

plot(xTs,sqrt(hhattestGJR), xlab="", ylab = "", main = "Wavelet")
lines(xTs,sqrt(predict(regmnewGJR,xtestGJR)),col="green")
dev.off()

#MSE and MAE calculation
GJRtestmse=c(
```

```
  sum((hhattestGJR-hgpreGJR)^2)/length(hhattestGJR),
  sum((predict(regmGJR,xtestGJR)-hhattestGJR)^2)/length(hhattestGJR),
  sum((predict(regmnewGJR,xtestGJR)-hhattestGJR)^2)/length(hhattestGJR),
  sum((predict(regmlin,xtestGJR)-hhattestGJR)^2)/length(hhattestGJR),
  sum((predict(regmpoly,xtestGJR)-hhattestGJR)^2)/length(hhattestGJR)
  )
GJRtestmse

GJRtestabs=c(
  sum(abs(hhattestGJR-hgpreGJR))/length(hhattestGJR),
  sum(abs(predict(regmGJR,xtestGJR)-hhattestGJR))/length(hhattestGJR),
  sum(abs(predict(regmnewGJR,xtestGJR)-hhattestGJR))/length(hhattestGJR),
  sum(abs(predict(regmlin,xtestGJR)-hhattestGJR))/length(hhattestGJR),
  sum(abs(predict(regmpoly,xtestGJR)-hhattestGJR))/length(hhattestGJR))
GJRtestabs


#TARCH
xtrainTARCH=cbind(u[3:((T-M)-1)], abs(u[3:((T-M)-1)]),hhat[3:((T-M)-1)]^0.5)
hhattrainTARCH=hhat[4:(T-M)]^0.5
xtestTARCH=cbind(u[((T-M)+1):(T-1)],abs(u[((T-M)+1):(T-1)]),hhat[((T-M)+1):(T-1)]^0.5)
hhattestTARCH=hhat[((T-M)+2):(T)]^0.5


tobjTARCH=tune.svm(x=xtrainTARCH,y=hhattrainTARCH,cost=c(2,5,10,25),epsilon=c(0.01,0.025,.05,.1)
tobjTARCH$best.parameters

CvalueTARCH=tobjTARCH$best.parameters$cost
epsvalueTARCH=tobjTARCH$best.parameters$epsilon

regmTARCH <-ksvm(xtrainTARCH,hhattrainTARCH,epsilon=epsvalueTARCH,C=CvalueTARCH,cross=5)
regmnewTARCH <-ksvm(xtrainTARCH,hhattrainTARCH,epsilon=epsvalueTARCH,kernel=k,C=CvalueTARCH,cros
regmlin <-ksvm(xtrainTARCH,hhattrainTARCH,epsilon=epsvalueTARCH,kernel="vanilladot",C=CvalueTARC
regmpoly <-ksvm(xtrainTARCH,hhattrainTARCH,epsilon=epsvalueTARCH,kernel="polydot",C=CvalueTARCH,

fitTARCH=garchFit(~ garch(1,1), data = utrain, delta=1, leverage=TRUE,include.mean=FALSE, trace
fitTARCH
volatilityTARCH = slot(fitTARCH, "h.t")

#MSE and MAE calculation
TARCHtrainmse=c(
  sum((volatilityTARCH[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2)^2)/(LTr-(BI-1)),
  sum((predict(regmTARCH,xtrainTARCH)[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2)^2)/(LTr-(BI-1)),
  sum((predict(regmnewTARCH,xtrainTARCH)[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2)^2)/(LTr-(BI-1)),
  sum((predict(regmlin,xtrainTARCH)[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2)^2)/(LTr-(BI-1)),
```

```
  sum((predict(regmpoly,xtrainTARCH)[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2)^2)/(LTr-(BI-1)))
TARCHtrainmse

TARCHtrainabs=c(
  sum(abs(volatilityTARCH[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2))/(LTr-(BI-1)),
  sum(abs(predict(regmTARCH,xtrainTARCH)[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2))/(LTr-(BI-1)),
  sum(abs(predict(regmnewTARCH,xtrainTARCH)[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2))/(LTr-(BI-1)),
  sum(abs(predict(regmlin,xtrainTARCH)[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2))/(LTr-(BI-1)),
  sum(abs(predict(regmpoly,xtrainTARCH)[BI:LTr]^2-hhattrainTARCH[BI:LTr]^2))/(LTr-(BI-1)))
TARCHtrainabs

#test data##
plot(xTs,hhattestTARCH^2,type="p")
lines(xTs,predict(regmTARCH,xtestTARCH)^2,col="red")

plot(xTs,hhattestTARCH^2,type="p")
lines(xTs,predict(regmnewTARCH,xtestTARCH)^2,col="green")

BTARCH <- coef(fitTARCH)
hgpreTARCH=numeric(length(hhattestTARCH))
hgpreTARCH[1]=hhattestTARCH[1]
for (i in 1:(length(hhattestTARCH)-1)){
  hgpreTARCH[i+1]=BTARCH[1]+BTARCH[2]*(abs(utest[i])-BTARCH[3]*utest[i])+BTARCH[4]*hgpreTARCH[i]
}

plot(xTs,hhattestTARCH^2,type="p")
lines(xTs,hgpreTARCH^2,col="blue")

#MSE and MAE calculation
TARCHtestmse=c(
  sum((hhattestTARCH^2-hgpreTARCH^2)^2)/length(hhattestTARCH),
  sum((predict(regmTARCH,xtestTARCH)^2-hhattestTARCH^2)^2)/length(hhattestTARCH),
  sum((predict(regmnewTARCH,xtestTARCH)^2-hhattestTARCH^2)^2)/length(hhattestTARCH),
  sum((predict(regmlin,xtestTARCH)^2-hhattestTARCH^2)^2)/length(hhattestTARCH),
  sum((predict(regmpoly,xtestTARCH)^2-hhattestTARCH^2)^2)/length(hhattestTARCH))
TARCHtestmse

TARCHtestabs=c(
  sum(abs(hhattestTARCH^2-hgpreTARCH^2))/length(hhattestTARCH),
  sum(abs(predict(regmTARCH,xtestTARCH)^2-hhattestTARCH^2))/length(hhattestTARCH),
  sum(abs(predict(regmnewTARCH,xtestTARCH)^2-hhattestTARCH^2))/length(hhattestTARCH),
  sum(abs(predict(regmlin,xtestTARCH)^2-hhattestTARCH^2))/length(hhattestTARCH),
  sum(abs(predict(regmpoly,xtestTARCH)^2-hhattestTARCH^2))/length(hhattestTARCH))

TARCHtestabs
```

```
#TSGARCH
xtrainTSGARCH=cbind(abs(u[3:((T-M)-1)]),hhat[3:((T-M)-1)]^0.5)
hhattrainTSGARCH=hhat[4:(T-M)]^0.5
xtestTSGARCH=cbind(abs(u[((T-M)+1):(T-1)]),hhat[((T-M)+1):(T-1)]^0.5)
hhattestTSGARCH=hhat[((T-M)+2):(T)]^0.5

tobjTSGARCH=tune.svm(x=xtrainTSGARCH,y=hhattrainTSGARCH,cost=c(2,5,10,25),epsilon=c(0.01,0.025,0
tobjTSGARCH$best.parameters

CvalueTSGARCH=tobjTSGARCH$best.parameters$cost
epsvalueTSGARCH=tobjTSGARCH$best.parameters$epsilon

regmTSGARCH <-ksvm(xtrainTSGARCH,hhattrainTSGARCH,epsilon=epsvalueTSGARCH,C=CvalueTSGARCH,cross=
regmnewTSGARCH <-ksvm(xtrainTSGARCH,hhattrainTSGARCH,epsilon=epsvalueTSGARCH,kernel=k,C=CvalueTS
regmlin <-ksvm(xtrainTSGARCH,hhattrainTSGARCH,epsilon=epsvalueTSGARCH,kernel="vanilladot",C=Cval
regmpoly <-ksvm(xtrainTSGARCH,hhattrainTSGARCH,epsilon=epsvalueTSGARCH,kernel="polydot",C=Cvalue

fitTSGARCH=garchFit(~ garch(1,1), data = utrain, delta=1, include.mean=FALSE, trace = FALSE)
fitTSGARCH
volatilityTSGARCH = slot(fitTSGARCH, "h.t")


par(mfrow=c(1,1))
plot(xTr,hhattrainTSGARCH^2,type="p")
lines(xTr,predict(regmTSGARCH,xtrainTSGARCH)^2,col="red")

plot(xTr,hhattrainTSGARCH^2,type="p")
lines(xTr,predict(regmnewTSGARCH,xtrainTSGARCH)^2,col="green")

plot(xTr,hhattrainTSGARCH^2,type="p")
lines(xTr,volatilityTSGARCH^2,col="blue")

BI=1

#MSE and MAE calculation
TSGARCHtrainmse=c(
  sum((volatilityTSGARCH[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2)^2)/(LTr-(BI-1)),
  sum((predict(regmTSGARCH,xtrainTSGARCH)[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2)^2)/(LTr-(BI-1)),
  sum((predict(regmnewTSGARCH,xtrainTSGARCH)[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2)^2)/(LTr-(BI-1
  sum((predict(regmlin,xtrainTSGARCH)[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2)^2)/(LTr-(BI-1)),
  sum((predict(regmpoly,xtrainTSGARCH)[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2)^2)/(LTr-(BI-1)))

TSGARCHtrainmse
```

```
TSGARCHtrainabs=c(
  sum(abs(volatilityTSGARCH[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2))/(LTr-(BI-1)),
  sum(abs(predict(regmTSGARCH,xtrainTSGARCH)[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2))/(LTr-(BI-1))
  sum(abs(predict(regmnewTSGARCH,xtrainTSGARCH)[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2))/(LTr-(BI-
  sum(abs(predict(regmlin,xtrainTSGARCH)[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2))/(LTr-(BI-1)),
  sum(abs(predict(regmpoly,xtrainTSGARCH)[BI:LTr]^2-hhattrainTSGARCH[BI:LTr]^2))/(LTr-(BI-1)))

TSGARCHtrainabs

#test data##
plot(xTs,hhattestTSGARCH^2,type="p")
lines(xTs,predict(regmTSGARCH,xtestTSGARCH)^2,col="red")

plot(xTs,hhattestTSGARCH^2,type="p")
lines(xTs,predict(regmnewTSGARCH,xtestTSGARCH)^2,col="green")

BTSGARCH <- coef(fitTSGARCH)
hgpreTSGARCH=numeric(length(hhattestTSGARCH))
hgpreTSGARCH[1]=hhattestTSGARCH[1]
for (i in 1:(length(hhattestTSGARCH)-1)){
  hgpreTSGARCH[i+1]=BTSGARCH[1]+BTSGARCH[2]*(abs(utest[i]))+BTSGARCH[3]*hgpreTSGARCH[i]
}

plot(xTs,hhattestTSGARCH^2,type="p")
lines(xTs,hgpreTSGARCH^2,col="blue")

#MSE and MAE calculation
TSGARCHtestmse=c(
  sum((hhattestTSGARCH^2-hgpreTSGARCH^2)^2)/length(hhattestTSGARCH),
  sum((predict(regmTSGARCH,xtestTSGARCH) ^2-hhattestTSGARCH^2)^2)/length(hhattestTSGARCH),
  sum((predict(regmnewTSGARCH,xtestTSGARCH) ^2-hhattestTSGARCH^2)^2)/length(hhattestTSGARCH),
  sum((predict(regmlin,xtestTSGARCH) ^2-hhattestTSGARCH^2)^2)/length(hhattestTSGARCH),
  sum((predict(regmpoly,xtestTSGARCH) ^2-hhattestTSGARCH^2)^2)/length(hhattestTSGARCH))
TSGARCHtestmse

TSGARCHtestabs=c(
  sum(abs(hhattestTSGARCH^2-hgpreTSGARCH^2))/length(hhattestTSGARCH),
  sum(abs(predict(regmTSGARCH,xtestTSGARCH)^2-hhattestTSGARCH^2))/length(hhattestTSGARCH),
  sum(abs(predict(regmnewTSGARCH,xtestTSGARCH)^2-hhattestTSGARCH^2))/length(hhattestTSGARCH),
  sum(abs(predict(regmlin,xtestTSGARCH)^2-hhattestTSGARCH^2))/length(hhattestTSGARCH),
  sum(abs(predict(regmpoly,xtestTSGARCH)^2-hhattestTSGARCH^2))/length(hhattestTSGARCH))

TSGARCHtestabs
```

```
trainmse <- round((rbind(garchtrainmse, GJRtrainmse, TARCHtrainmse,
                         TSGARCHtrainmse)),5)

colnames(trainmse)=c("Model","Gauss","Wavelet", "Lin", "Poly")
rownames(trainmse)=c("GARCH","GJR","TARCH","TSGARCH")


testmse <- round((rbind(garchtestmse,GJRtestmse,TARCHtestmse,
                        TSGARCHtestmse)),5)

colnames(testmse)=c("Model","Gauss","Wavelet","Lin", "Poly")
rownames(testmse)=c("GARCH","GJR","TARCH","TSGARCH")
trainmse #Table of training MSE
testmse #Table of test MSE


trainabs <- round((rbind(garchtrainabs, GJRtrainabs,TARCHtrainabs,
                         TSGARCHtrainabs)),5)

colnames(trainabs)=c("Model","Gauss","Wavelet","Lin", "Poly")
rownames(trainabs)=c("GARCH","GJR","TARCH","TSGARCH")


testabs <- round((rbind(garchtestabs,GJRtestabs,TARCHtestabs,
                        TSGARCHtestabs)),5)

colnames(testabs)=c("Model","GAUSS","Wavelet","Lin", "Poly")
rownames(testabs)=c("GARCH","GJR","TARCH","TSGARCH")
trainabs #Table of training MAE
testabs #Table of test MAE


GKSN=cbind(nSV(regm),nSV(regmGJR),nSV(regmTARCH),nSV(regmTSGARCH))
WKSN=cbind(nSV(regmnew),nSV(regmnewGJR),nSV(regmnewTARCH),nSV(regmnewTSGARCH))

NSV=rbind(GKSN,WKSN)

colnames(NSV)=c("GARCH","GJR","TARCH","TSGARCH")
rownames(NSV)=c("Gaussian", "Wavelet")
print(NSV) #Table of number of support vectors
```