Forecasting EUR/USD Volatility Using Supervised Learning

Capstone Project – HarvardX Data Science Program

Christian Satzky

13 November 2020

Contents

1 Acknowledgement **3**

2 Introduction and Objective **3**

3 Executive Summary **3**

4 Modeling Approach **4**

5 Variables in the EUR/USD Dataset **4**

6 Data Exploration **5**
6.1 Summary Statistics . . . 5
6.2 Relationship Across Variables . . . 6
6.3 Stratification by Day of the Week . . . 7

7 Modeling **8**
7.1 The Benchmark Model . . . 9
7.2 Linear Model . . . 9
7.2.1 HAR-RV Model . . . 9
7.2.2 Weekday Effect Model . . . 9
7.3 Non-Linear Methods . . . 10
7.4 Tree Ensembles . . . 10
7.4.1 Random Forest . . . 10
7.4.2 Gradient Boosted Trees (XGBoost) . . . 11
7.5 Support Vector Machine (SVM) . . . 12
7.5.1 Linear Kernel . . . 12
7.5.2 Radial Basis Function . . . 13
7.6 K Nearest Neighbor . . . 14

8 Validation Set Performance **15**
8.1 Predictions During Pre-Crisis Period (January 2020) . . . 18
8.2 Predictions During Covid-19 Crisis (March 2020) . . . 19

9 Conclusion **19**

10 Outlook and Potential Improvements **20**

11 References **21**

12 Appendix 1: Financial Volatility **22**
12.1 Conditional Heteroskedascity . . . 22

13 Appendix 2: GARCH(1,1) Volatility Forecast **23**
13.1 GARCH(1,1) Out-of-Sample Performance . . . 24

1 Acknowledgement

This analysis is for partly fulfillment of the 'Capstone' module in the HarvardX Data Science program, an online learning initiative by Harvard University. The programming parts are written in R (R Core Team 2020).

All code is *fully reproducible* and all steps that led to the results are documented below. For the complete code please look into the *.Rmd file directly. In the rendered PDF version, most chunks of code are not displayed for an improved reading experience.

2 Introduction and Objective

This project deals with predicting EUR/USD volatility using supervised learning. For those unfamiliar with the topic of financial volatility and/or its measurement, I am providing an excursion in the appendix which you may like to read before continuing with this paper.

Accurate forecasting of volatility is crucial for the pricing of financial derivatives, portfolio allocation and effective risk management. The goal of this project is to find machine-learning methods that are best suited for volatility forecasting.

For this project, SHARELAB LIMITED provided data containing daily *realized volatility* (RV) of EUR/USD currency quotes. The data consists of a `training` set (data from 1/1/2010 to 12/31/2019), and a `validation` set (data from 1/1/2020 to 7/31/2020). The objective of this project is to predict RV_{t+1} (i.e. EUR/USD realized volatility for date $t+1$) as accurately as possible on the pseudo out-of-sample `validation` set.

Any analysis and modeling is strictly performed on the `training` dataset, while the `validation` subset is used only to assess the final model's performance per machine-learning method considered. The performance measure is the root mean square error (RMSE).

With consent of SHARELAB LIMITED, the data is publicly available in the *.Rdata file on GitHub.

3 Executive Summary

The dataset contains future realized volatility RV_{t+1} (y-variable) and predictors that are aggregates of *past* volatility. Data exploration reveals that (1) the relationship between the X variables and RV_{t+1} appears to be well approximated by a straight line and (2) RV seems to exhibit significant dependence with regards to day of the week.

When it comes to modeling, four general methods are considered: Linear regression, tree ensembles, support vector machine and k-nearest neighbor. The best performing model variants for each category are:

- Linear regression: 'Weekday effect model'
- Tree ensemble: Gradient boosted trees
- Support vector machine (SVM): SVM with linear kernel
- K-nearest neighbor (only one model considered)

Of all methods and models considered, the gradient boosted tree algorithm performs best with an RMSE of 0.0016 on the `validation` set. However, the `validation` set includes a period of market turmoil due to the Covid-19 pandemic that greatly impacted markets in March 2020. Rigorous analysis of the results shows that;

- During *market turmoil*, the gradient boosted trees (GBT) algorithm performs best
- During *normal* times, the support vector machine (SVM) algorithm with linear kernel performs best

The linear weekday effect model performs second best in each scenario and appears to be most robust to changing market conditions.

Noteworthy, in the appendix, it is found that the industry-prevalent GARCH(1,1) model is outperformed by all models considered.

4 Modeling Approach

To prevent potential overtraining, the `training` dataset is first split into `train` and `test` subsets. The `train` subset is explored using summary statistics and data visualization. In the process, some variables are transformed and additional predictors are created.

Secondly, different machine-learning methods are applied to estimate the conditional expectation $E[\text{RV}_{t+1}|X]$. For each method, any model variants and any tuning of parameters is done using the `train` and `test` sets only.

Finally, using the RMSE information on the `test` subset, *one* model *per algorithm* is chosen to be fit on the complete `training` set and evaluated on the `validation` set. Considering the RMSE performance on the `validation` set of each method's final model, the suitability of different algorithms to forecast volatility for the EUR/USD currency is discussed.

5 Variables in the EUR/USD Dataset

Prior to data exploration, below I am giving an overview regarding the meaning, calculation and interpretation of the provided variables in the EUR/USD realized volatility dataset.

```
# show variable names
names(training)
```

```
## [1] "date"  "rv_t1" "rv_d"  "rv_w"  "rv_m"
```

There are 5 variables provided:

1. 'date'
2. RV_{t+1} ('rv_t1')
3. RV^d ('rv_d')
4. RV^w ('rv_w')
5. RV^m ('rv_m')

RV is short for 'realized volatility'. Following Corsi (2009), RV for the EUR/USD exchange rate is calculated by SHARELAB LIMITED as follows:

$$RV^d = \sqrt{\sum_{m=0}^{M-1} r^2_{t-\frac{m}{M}}}$$

where:

RV^d: is the one-day realized volatility for the EUR/USD quote

r_t: is the intradaily return of the EUR/USD quote, $r_t = ln\frac{P_t}{P_{t-\frac{1}{M}}}$

M: is the number of equally spaced returns per day, here $M = 288$

ln: is the natural logarithm

P_t: is the EUR/USD quote at time t

As seen above, the daily realized variance is the sum of *intradaily*, continuously compounded, squared returns. Note that there are $M = 288$ returns per day, implying that intradaily returns are observed at the 5-minute interval for the 24-hour EUR/USD trading day.

Both RV^w and RV^m are rolling, historical averages of RV^d. More precisely, RV^w is the *weekly* average of RV^d and RV^m is the *monthly* average of RV^d. Note that for the EUR/USD currency pair, there are 5 trading days per week and roughly 22 trading days a month.

$$RV^w = \frac{1}{5}\sum_{j=0}^{4} RV^d_{t-j}$$

$$RV^m = \frac{1}{22}\sum_{j=0}^{21} RV^d_{t-j}$$

where:

t: current trading day (as shown in the variable 'date')

$t-j$: j^{th} trading day before t

Note that RV_{t+1} is the 'Y' variable we are trying to predict. RV_{t+1} is the one-day ahead future value of daily realized volatility, RV^d.

In conclusion, in this project we are tying to predict the next trading day's EUR/USD volatility, RV_{t+1} given historical volatility measures RV^d, RV^w and RV^m. For more information see Corsi, 2009.

6 Data Exploration

Data exploration and any modeling is strictly performed on the `train` set, which contains 90% of the observations given in the `training` dataset.

```
# split 'training' into 'train' and 'test' sets

# order 'training' set by date ascending
training <- training[order(date), ]

# pick 10% of (future) time-series for 'test' set
dates_test_set <- tail(training$date, round(0.1 * nrow(training)))
test <- training[date %in% dates_test_set, ]
train <- training[!date %in% dates_test_set, ]
```

6.1 Summary Statistics

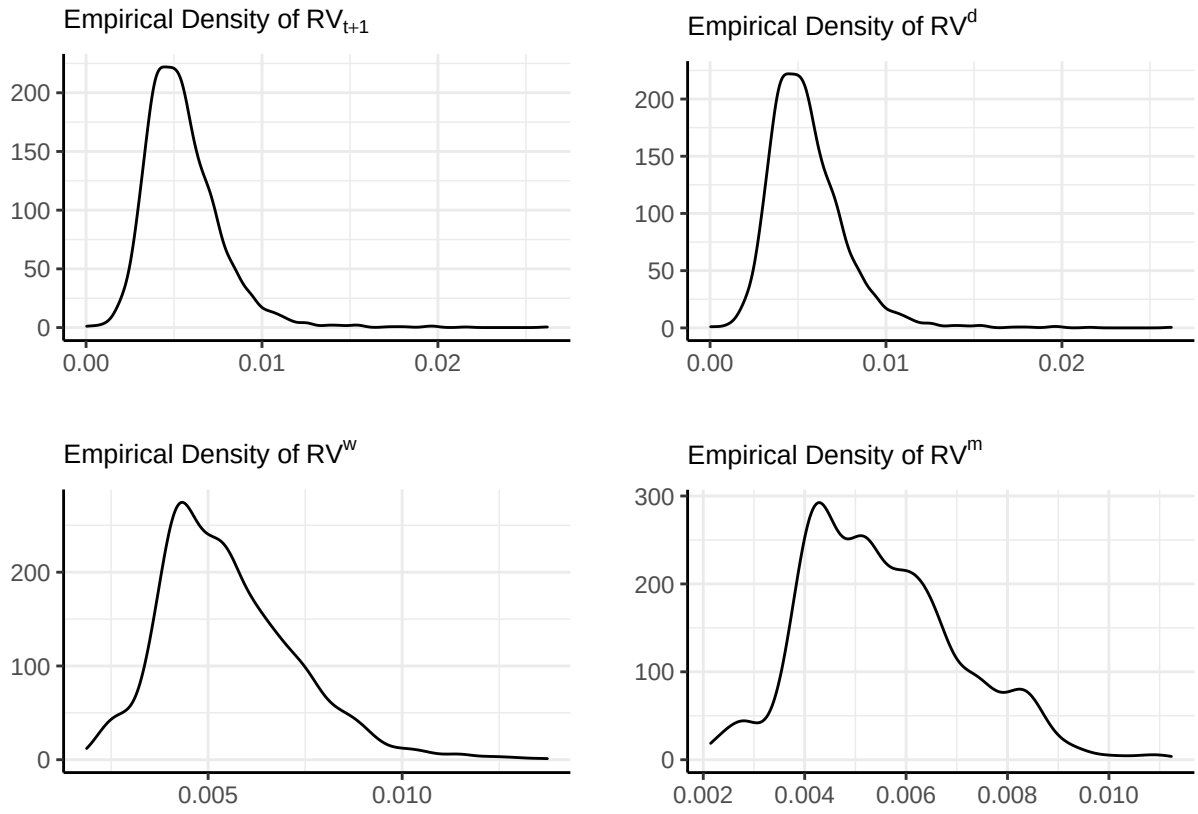
First, I am describing the provided variables for the `train` partition using summary statistics.

	date	RV_{t+1}	RV^d	RV^w	RV^m
class	Date	numeric	numeric	numeric	numeric
min	2010-01-01	3e-05	3e-05	0.00187	0.00215
max	2018-12-31	0.02622	0.02622	0.01374	0.01123
mean		0.00552	0.00552	0.00552	0.00553
median		0.00518	0.00518	0.00527	0.00528
sd		0.00219	0.00219	0.00176	0.00155

From the table above, we observe:

- the time-series for the `train` partition spawns from 1 January 2010 to 31 December 2018
- the `date` variable is correctly formatted as `Date` and the other variables are all `numeric`
- all variables are positive (the lowest value of RV^d is 0.00003). This is expected as volatility is proxied by the standard deviation of the return time-series. Therefore, volatility must be greater than 0 as long as the EUR/USD returns are not constant
- the mean value is approximately the same across daily, weekly and monthly aggregations of RV

- across all RV variables, the median value is slightly lower than the mean. This implies that the distribution for all measures of RV is skewed to the right
- note that $sd(RV_{t+1}) = sd(RV^d) > sd(RV^w) > sd(RV^m)$. This is expected as the monthly or weekly measures of RV are computed from more observations than the daily measure, and RV^d is hence noisier than RV^w and RV^m



The empirical densities are in line with the fact that RV^w and RV^m are cumulative aggregations of RV^d. Rare, extreme events with high volatility have greater persistence on weekly and even more so on monthly measures of volatility.

6.2 Relationship Across Variables

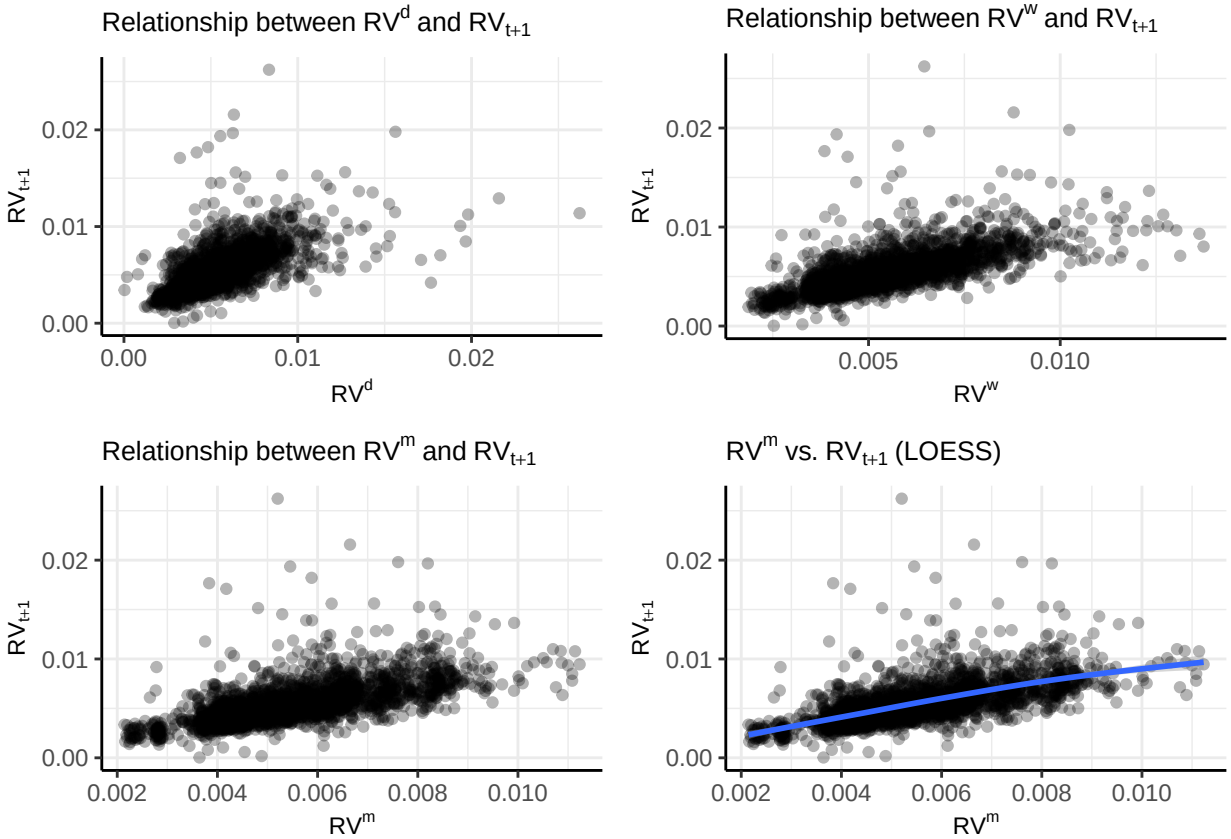
As RV^w has overlapping information with RV^d and RV^m has overlapping information with both RV^w and RV^d, it is straightforward to assume that all X variables are positively correlated. This is confirmed by the correlation matrix below:

```
# compute Pearson correlation of features
corr <- round(cor(train[, .(rv_t1, rv_d, rv_w, rv_m)], method = "pearson"), 2)
var_names <- c("$RV_{t+1}$", "$RV^d$", "$RV^w$", "$RV^m$")
row.names(corr) <- var_names
colnames(corr) <- var_names

# display latex-style table
kable(corr, escape = FALSE, booktabs = TRUE) %>% kable_styling(position = "center")
```

	RV_{t+1}	RV^d	RV^w	RV^m
RV_{t+1}	1.00	0.62	0.67	0.63
RV^d	0.62	1.00	0.79	0.67
RV^w	0.67	0.79	1.00	0.86
RV^m	0.63	0.67	0.86	1.00

Looking at the scatter plots of the X variables vs. RV_{t+1} provides greater detail regarding the variables' relationship:



As seen in the plots above, the relationship between the independent variables vs. RV_{t+1} appears to be well approximated by a straight line. Hence, a linear model to estimate $E[\mathrm{RV}_{t+1}|X]$ might performs well.

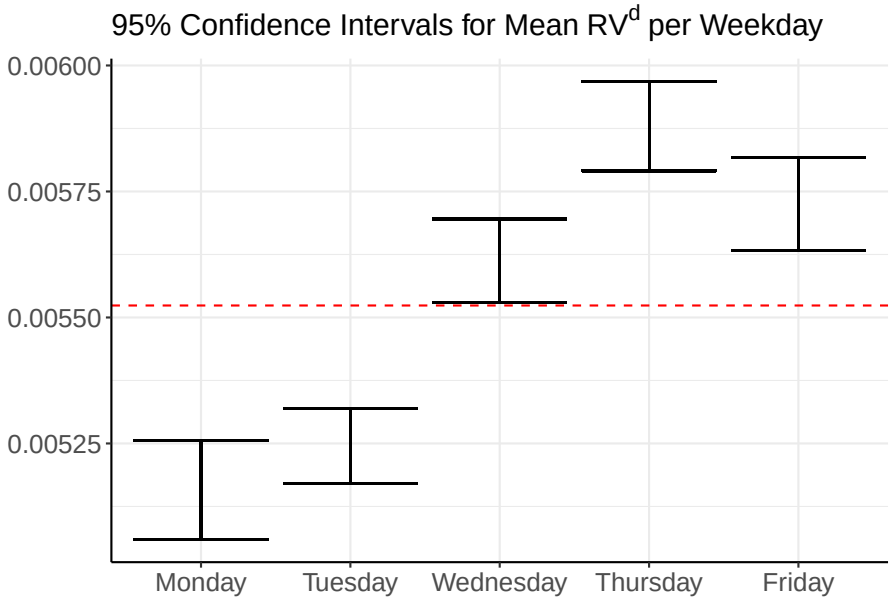
6.3 Stratification by Day of the Week

The EUR/USD currency pair is traded from Mondays to Fridays. When information accumulates *over the weekends*, it can only impact volatility on the following Monday. Likewise, if market participants are required to close their positions *before* the start of the weekend, this might impact Friday's volatility. Hence, it is sensible to investigate volatility dependencies with respect to days of the week.

First, the variable `date` is used to extract `weekday`.

```
# extract 'weekday' from 'date' variable
train[, `:=`(weekday, weekdays(date))]
```

Secondly, I am plotting 95% confidence intervals for the *mean value* of RV^d per `weekday`. Note that stratifying RV^d is equivalent to stratifying RV_{t+1}, but for the latter the `weekday` is shifted by one day.



In the above plot, the dashed red line marks the overall mean of RV^d. Surprisingly, the 95% confidence interval for $\overline{RV^d}$ is *lowest* on Mondays. It might be the case that on average, information flow on weekends is modest and risk-averse market participants close positions on Fridays, when volatility is *greater* than on the average weekday.

Note that *none* of the 95% confidence intervals overlaps with the overall average value of 0.00552. This is reasonable evidence to assume some variability across weekdays and hence I will include the newly created `weekday`-variable for modeling via the creation of dummy variables.

$$D^i = \begin{cases} 1, & \text{if weekday} = i \\ 0, & \text{otherwise} \end{cases}$$

where:

D: Dummy variable, $D : \Omega \to \{0, 1\}$

i: Value of `weekday`, $i \in \{\text{Monday}, \text{Tuesday}, \text{Wednesday}, \text{Thursday}, \text{Friday}\}$

To avoid perfect multicollinearity, it is necessary to only create dummy variables for $N - 1$ categories of `weekday`. Hence, I am only including dummy variables for Monday to Thursday. The library `fastDummies` (Kaplan and Schlegel 2020) provides a convenient function for dummy variable extraction.

```
# create dummy variables on 'train' set
train <- dummy_cols(.data = train, select_columns = "weekday")

# exclude Friday's dummy variable
train[, ':='(weekday_Friday, NULL)]
```

7 Modeling

Now that the data is sufficiently explored and additional features are created, in this section I am fitting different models on the `train` set and check for an RMSE estimate on the `test` set.

7.1 The Benchmark Model

The benchmark model is simply the average value of RV_{t+1}, without using any of the other variables. It can be directly evaluated on the `test` set.

$$RV_{t+1} = \mu + \epsilon_{t+1}$$

The benchmark model achieves an RMSE of 0.00249 on the `test` set.

7.2 Linear Model

Other features in the dataset are RV^d, RV^w, RV^m and the newly created dummy variables D^i. In the following, I am incorporating these variables using different model variants.

7.2.1 HAR-RV Model

For the first linear model variant, I am including all features except for the `weekday` dummy variables. This model is equivalent to Corsi's (2009) Heterogeneous Autoregressive model of Realized Volatility (HAR-RV).

$$RV_{t+1} = \alpha + \beta^d RV^d + \beta^w RV^w + \beta^m RV^m + \epsilon_{t+1}$$

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.0004832	0.0001212	3.987924	6.87e-05
rv_d	0.2421466	0.0241954	10.007983	0.00e+00
rv_w	0.3644108	0.0435179	8.373823	0.00e+00
rv_m	0.3055065	0.0410421	7.443731	0.00e+00

As seen above, all coefficient estimates in the HAR-RV model are significantly different from zero. Also note that all coefficients are positive, which is in line with what was discovered in the Relationship Across Variables section. Below, I am summarizing the model's RMSE performance on the `test` set.

Model	Variables	$\text{RMSE}_{\text{test}}$
Benchmark	μ	0.0024906
Linear Model		
HAR-RV	α, RV^d, RV^w, RV^m	**0.0008829**

The HAR-RV model greatly outperforms the benchmark model. The RMSE on the `test` set improves by 64.55%.

7.2.2 Weekday Effect Model

For the second linear model I am adding *all* features, including dummy variables D^i for the newly created `weekday` variable. More specifically, for 5 workdays, 4 dummy variables are introduced to capture time dependencies across weekdays as shown in the data exploration section.

$$RV_{t+1} = \alpha + \beta^d RV^d + \beta^w RV^w + \beta^m RV^m + \sum \beta_i D_i + \epsilon_{t+1}$$

where D is a dummy variable, and $i \in$ [Monday, Tuesday, Wednesday, Thursday].

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.0000562	0.0001363	0.412543	0.6799793
rv_d	0.2409071	0.0243803	9.881206	0.0000000
rv_w	0.3620415	0.0432806	8.364991	0.0000000
rv_m	0.3101740	0.0404767	7.663021	0.0000000
weekday_Monday	0.0002241	0.0001025	2.187635	0.0287949
weekday_Tuesday	0.0005862	0.0001023	5.732597	0.0000000
weekday_Wednesday	0.0007588	0.0001016	7.468421	0.0000000
weekday_Thursday	0.0005378	0.0001016	5.291782	0.0000001

Each dummy variable has a parameter value that is significantly different from zero (pval < 0.05). Note that after accounting for the variability across `weekday`, the intercept of the model, α is *not* significantly different from zero.

Model	Variables	$\text{RMSE}_{\text{test}}$
Benchmark	μ	0.0024906
Linear Model		
HAR-RV	α, RV^d, RV^w, RV^m	0.0008829
Weekday Effect	$\alpha, RV^d, RV^w, RV^m, \sum D_i$	**0.0008568**

As summarized in the table above, the weekday effect model improves RMSE performance in comparison to the HAR-RV model by 2.96%.

7.3 Non-Linear Methods

From this section onward, I am considering more 'versatile' methods as opposed to linear regression. The advantage is that non-parametric, machine-learning models impose "no or very limited" assumptions on the data (Whitley and Ball 2002) and provide greater flexibility in how the parameters can be used.

As noted by Irizarry, Rafael A. (2020), the `caret`-package (Kuhn, M. et al 2020) provides a convenient way to train machine-learning algorithms using cross-validation (CV). Furthermore, the package standardizes R functions across several libraries. In the following, I am using `caret` to tune and fit more flexible algorithms.

7.4 Tree Ensembles

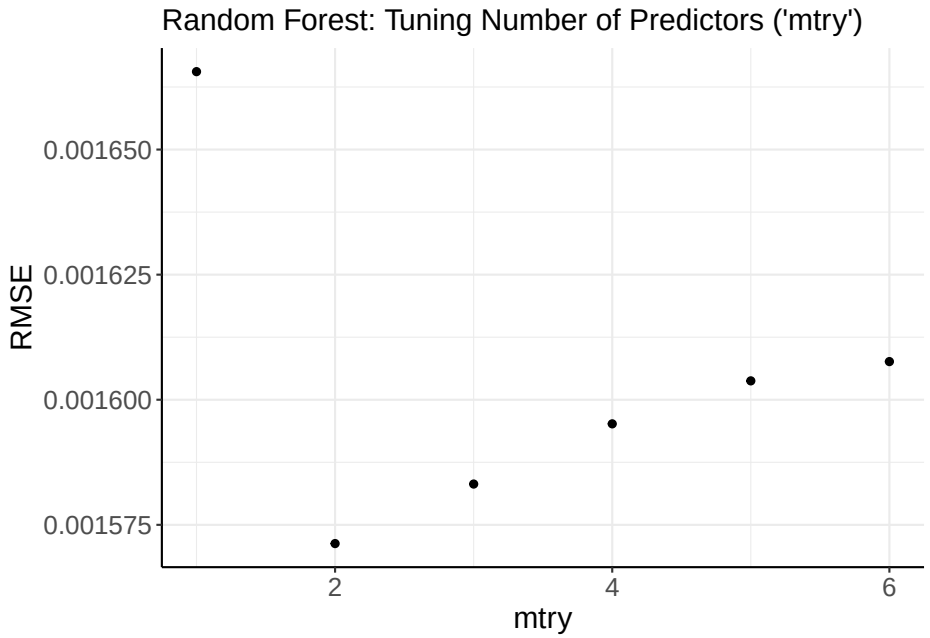
In this section I am considering tree ensembles, specifically random forests (RF) and gradient boosted trees (GBT).

7.4.1 Random Forest

The random forest algorithm is a method that fits many decision trees with randomly selected predictors and aggregates the resulting trees to form a prediction (bootstrap aggregating). This procedure reduces the instability of single regression trees (Irizarry, Rafael A. 2020).

The argument `getModelInfo('rf')$rf$parameters` reveals that the random forest algorithm of the package `randomForest` (Breiman, L. et al 2018) only has one tuning parameter, i.e. `mtry`. As mentioned above, for each decision tree that is fitted, random forest *randomly* selects the predictors to be used. `mtry` sets the *number* of features to be picked at random for each tree. As there are 7 independent variables available, it is most sensible to consider a value for `mtry`< 7. Using 10-fold cross-validation, I am tuning `mtry` on the `train` set. I am considering values `mtry`$\in \{1, 2, ..., 6\}$.

The random forest algorithm to be used is developed by Breiman et all, 2018 and is available in the `randomForest`-library.



Observe that the random forest model is best performing for `mtry`= 2. Using this parameter value, I am fitting the random forest model on the `train` set and summarize the RMSE performance of predictions on the `test` set.

Model	Variables	$\text{RMSE}_{\text{test}}$
Benchmark	μ	0.0024906
Linear Model		
HAR-RV	α, RV^d, RV^w, RV^m	0.0008829
Weekday Effect	$\alpha, RV^d, RV^w, RV^m, \sum D_i$	**0.0008568**
Tree Ensembles		
Random Forest	$RV^d, RV^w, RV^m, \sum D_i$; mtry = 2	0.0009372

The well-tuned random forest model performs much better than the benchmark model, but fails to achieve an RMSE close to the performance of the linear regression variants. As noted in the data exploration section, the relationship of X vs. RV_{t+1} appears to be well-approximated by a straight line. There seem to be limited non-linear dependencies that random forest could take advantage of.

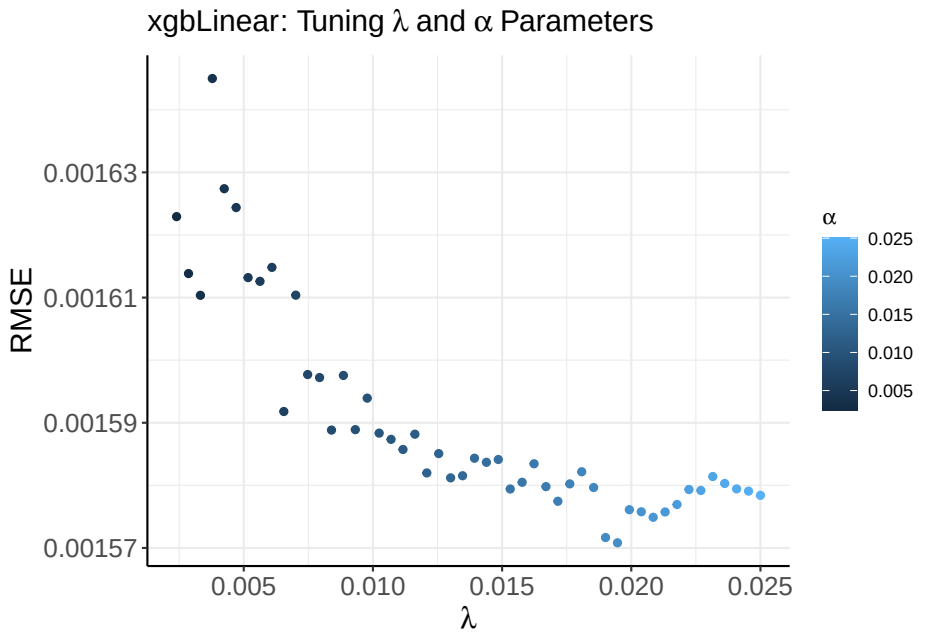
7.4.2 Gradient Boosted Trees (XGBoost)

While the random forest algorithm uses grown decision trees, 'boosting' is based on shallow trees. While the random forest algorithm aims at *reducing variance* by aggregating many, fully grown decision trees, gradient boosting aims at reducing bias of shallow trees.

The XGBoost package (He, T. et al 2020) has the following tuning parameters:

- `eta`: Shrinkage of feature weights. Default value 0.3, eta $\in [0, 1]$
- `lambda` (λ): L2 regularization term on weights, default value 1
- `alpha` (α): L1 regularization term on weights, default value 0
- `nrounds`: 50~150

Going forward, I am accepting the default values for `eta` and `nrounds` and tune the α and λ parameters.



As seen in the plot above, the lowest cross-validated RMSE on the `train` set is achieved for $\lambda = \alpha = 0.019$. Analog to the previous section, I am applying the model on the out-of-sample `test` set.

Model	Variables	$\text{RMSE}_{\text{test}}$
Benchmark	μ	0.0024906
Linear Model		
HAR-RV	α, RV^d, RV^w, RV^m	0.0008829
Weekday Effect	$\alpha, RV^d, RV^w, RV^m, \sum D_i$	**0.0008568**
Tree Ensembles		
Random Forest	$RV^d, RV^w, RV^m, \sum D_i$; mtry $= 2$	0.0009372
Gradient Boosted Trees	$RV^d, RV^w, RV^m, \sum D_i; \alpha = \lambda = 0.019$	0.0008966

Gradient boosting significantly outperforms random forest in terms of `test` set's RMSE. Still, it fails to achieve the performance of the linear model variants.

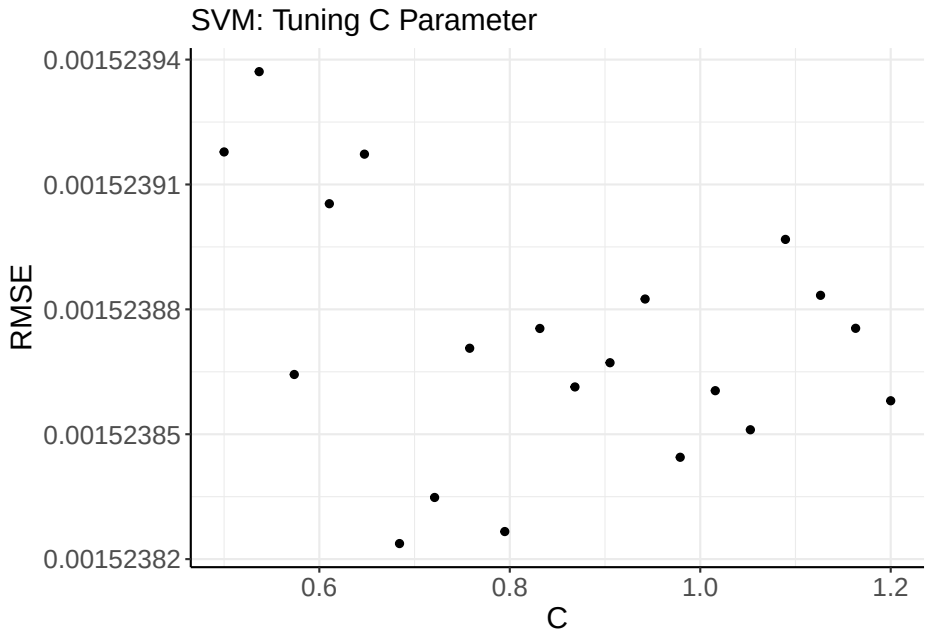
7.5 Support Vector Machine (SVM)

The idea of a support vector machine algorithm is to separate those points that are hard to distinguish (i.e. points that are closest to one another). With this separation in place, points that are further away from one another are more likely to be correctly classified.

Applying the SVM algorithm to many predictors can be achieved efficiently by using a kernel function. Using the `kernlab` library (Karatzoglou, A. et al 2019), I am considering a linear kernel function and the more flexible radial basis kernel.

7.5.1 Linear Kernel

For the SVM algorithm with linear kernel, there is only one parameter to tune, i.e. `C` (cost). The cost parameter determines the margin of the separating hyperplane. A low value of `C` is associated with a large margin, a high value is associated with a small margin. Going forward, I am considering values `C`$\in [0.5, 1.2]$.



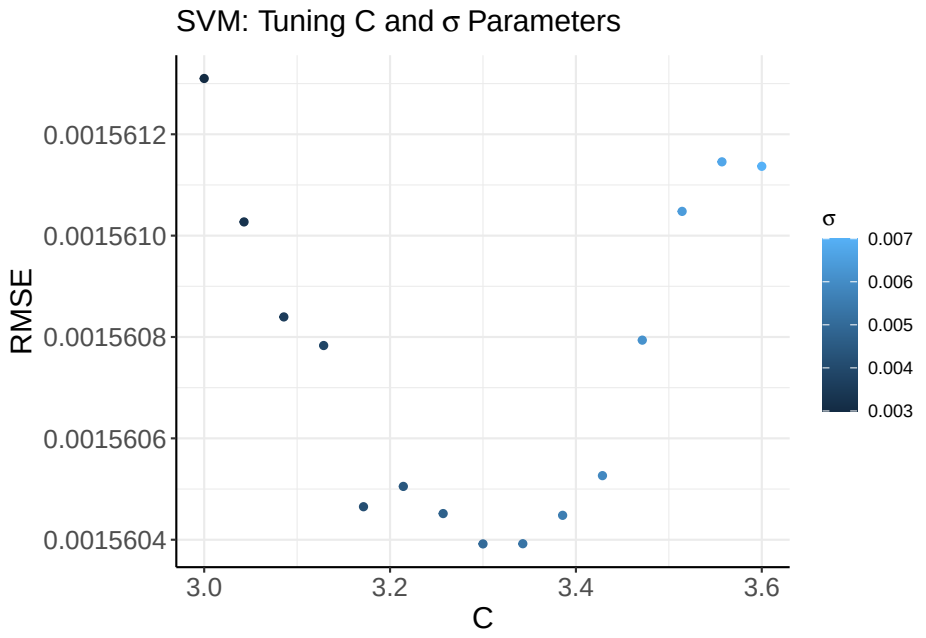
According to the 25-fold cross validation procedure on the `train` set, a parameter value around 0.7 seems to be the optimal choice (`C`= 0.6842).

Model	Variables	$\text{RMSE}_{\text{test}}$
Benchmark	μ	0.0024906
Linear Model		
HAR-RV	α, RV^d, RV^w, RV^m	0.0008829
Weekday Effect	$\alpha, RV^d, RV^w, RV^m, \sum D_i$	0.0008568
Tree Ensembles		
Random Forest	$RV^d, RV^w, RV^m, \sum D_i$; mtry = 2	0.0009372
Gradient Boosted Trees	$RV^d, RV^w, RV^m, \sum D_i; \alpha = \lambda = 0.019$	0.0008966
Support Vector Machine		
SVM Linear Kernel	$RV^d, RV^w, RV^m, \sum D_i; C = 0.6842$	**0.0008178**

So far, the SVM with linear kernel achieves the best RMSE performance on the out-of-sample `test` set. It beats the second best weekday effect model's RMSE by 4.77%.

7.5.2 Radial Basis Function

Below, I am tuning a SVM with the more flexible radial basis kernel. For the parameter values to be tuned, `C` and `sigma` (σ). For the tuning grid I am considering values `C`$\in [3, 3.6]$ and $\sigma \in [0.003, 0.007]$.



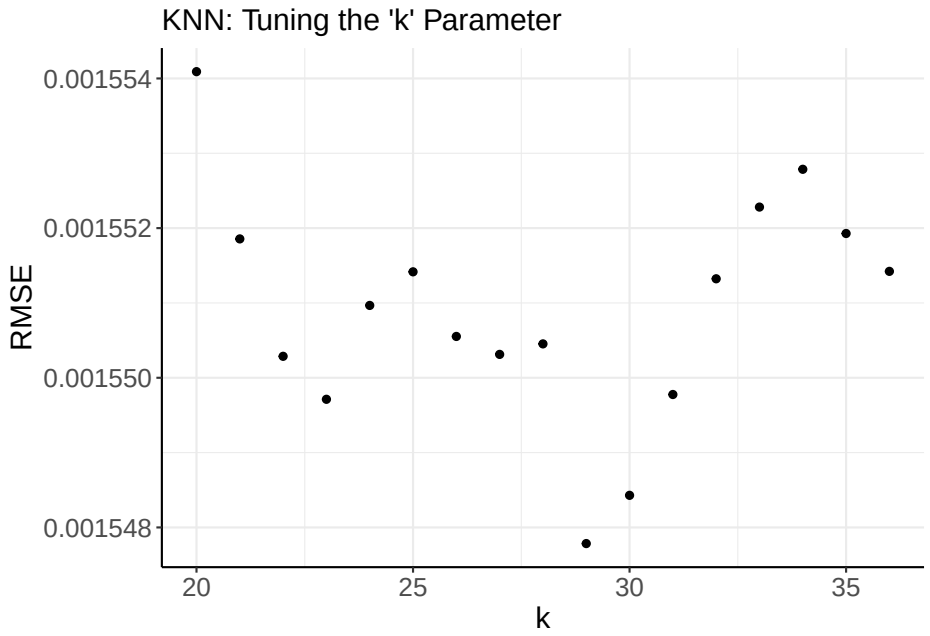
The optimal parameter values according to 10-fold cross-validation on the `train` set are `C`$= 3.3$ and $\sigma = 0.005$.

Model	Variables	$\text{RMSE}_{\text{test}}$
Benchmark	μ	0.0024906
Linear Model		
HAR-RV	α, RV^d, RV^w, RV^m	0.0008829
Weekday Effect	$\alpha, RV^d, RV^w, RV^m, \sum D_i$	0.0008568
Tree Ensembles		
Random Forest	$RV^d, RV^w, RV^m, \sum D_i$; mtry $= 2$	0.0009372
Gradient Boosted Trees	$RV^d, RV^w, RV^m, \sum D_i; \alpha = \lambda = 0.019$	0.0008966
Support Vector Machine		
SVM Linear Kernel	$RV^d, RV^w, RV^m, \sum D_i; C = 0.6842$	**0.0008178**
SVM Radial Basis	$RV^d, RV^w, RV^m, \sum D_i; C = 3.3, \sigma = 0.005$	0.000819

Even after excessive tuning, the radial basis function kernel of the SVM algorithm fails to outperform the linear kernel in the out-of-sample application. This is consistent with the discovery that less flexible methods might be superior for the dataset at hand.

7.6 K Nearest Neighbor

The k nearest neighbor algorithm classifies a data point by looking at the outcomes of the k *closest* points to it. For regression, the mean y-variable value of the k closest points is assigned. For the tuning parameter `k` I am considering values `k`$\in [20, 36]$.



According to 20-fold cross validation on the `train` set, the optimal value for `k` is 29.

Model	Variables	$\text{RMSE}_{\text{test}}$
Benchmark	μ	0.0024906
Linear Model		
HAR-RV	α, RV^d, RV^w, RV^m	0.0008829
Weekday Effect	$\alpha, RV^d, RV^w, RV^m, \sum D_i$	0.0008568
Tree Ensembles		
Random Forest	$RV^d, RV^w, RV^m, \sum D_i$; mtry $= 2$	0.0009372
Gradient Boosted Trees	$RV^d, RV^w, RV^m, \sum D_i; \alpha = \lambda = 0.019$	0.0008966
Support Vector Machine		
SVM Linear Kernel	$RV^d, RV^w, RV^m, \sum D_i; C = 0.6842$	**0.0008178**
SVM Radial Basis	$RV^d, RV^w, RV^m, \sum D_i; C = 3.3, \sigma = 0.005$	0.000819
K Nearest Neighbor		
KNN	$RV^d, RV^w, RV^m, \sum D_i; k = 29$	0.0009446

When it comes to out-of-sample performance on the `test` set, the KNN algorithm performs worst. In comparison to the support vector machine with linear kernel the RMSE performance is 15.51% worse.

8 Validation Set Performance

The table above summarizes the `test` set's RMSE performance of all models considered. The best performing models *per algorithm* are:

- Linear model: Weekday effect model, $\text{RMSE}_{\text{test}} = 0.000857$
- Tree ensembles: Gradient boosted trees, $\text{RMSE}_{\text{test}} = 0.000897$
- Support vector machine: SVM with linear kernel, $\text{RMSE}_{\text{test}} = \mathbf{0.000818}$
- K Nearest Neighbor (only one model considered), $\text{RMSE}_{\text{test}} = 0.000945$

Going forward, these four models are fit to the *complete* `training` set and evaluated on the `validation` set. For tuning on the `training` set, for each model, I am considering a grid range of $\pm 50\%$ of the previously

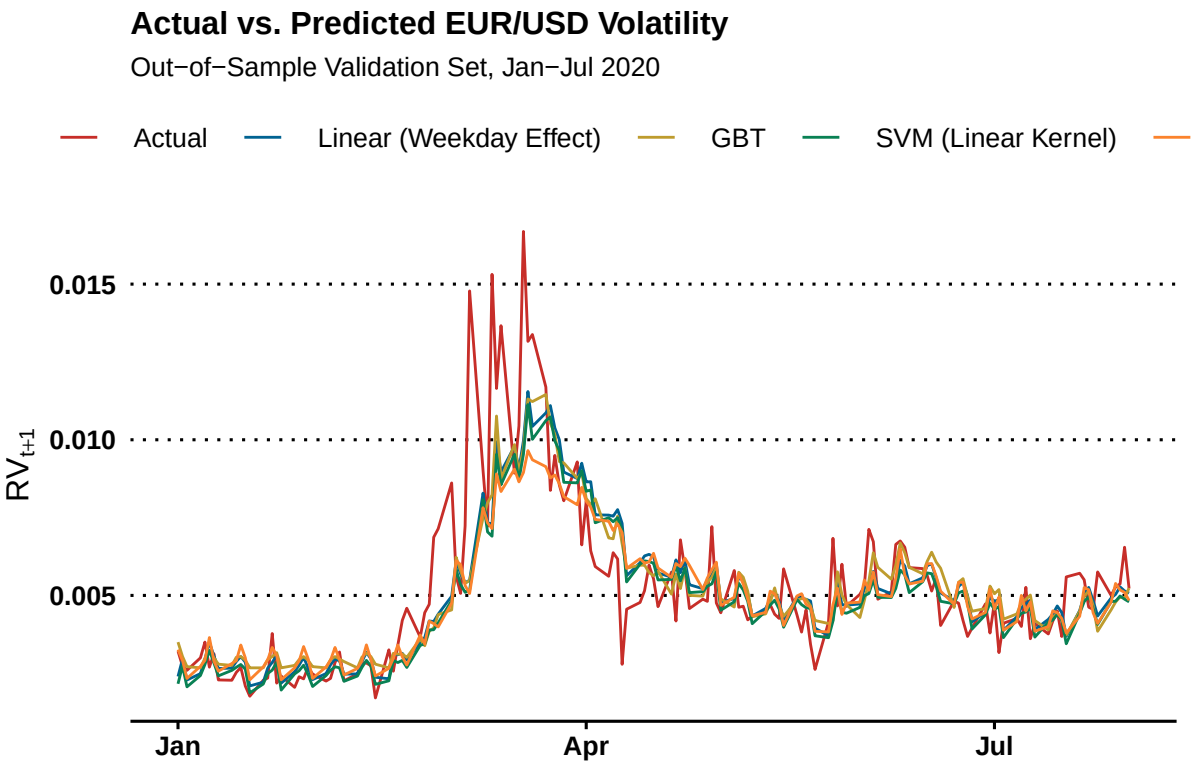
obtained parameter values:

Model	$\text{Parameter}_{\text{train}}$	$\pm 50\%\text{Parameter Range}_{\text{Validation}}$
Weekday Effect		
Gradient Boosted Trees	$\sigma = \lambda = 0.019465$	$[0.0097325, 0.0291975]$
SVM (Linear Kernel)	$C = 0.684211$	$[0.342105, 1.026315]$
KNN	$k = 29$	$[15, 44]$

For each model with parameters to be trained, the above parameter range $[a, b]$ is considered using 10-fold cross validation and a grid length of 15. This can be achieved using the `seq()`-function, e.g. for the SVM (linear kernel) `seq(0.342105, 1.026315, length.out = 15)`.

Model	Variables	$\text{RMSE}_{\text{test}}$	$\text{RMSE}_{\text{validation}}$
Benchmark	μ	0.0024906	
Linear Model			
HAR-RV	α, RV^d, RV^w, RV^m	0.0008829	
Weekday Effect	$\alpha, RV^d, RV^w, RV^m, \sum D_i$	0.0008568	0.0016219
Tree Ensembles			
Random Forest	$RV^d, RV^w, RV^m, \sum D_i$; mtry $= 2$	0.0009372	
Gradient Boosted Trees	$RV^d, RV^w, RV^m, \sum D_i; \alpha = \lambda = 0.019$	0.0008966	**0.0015951**
Support Vector Machine			
SVM Linear Kernel	$RV^d, RV^w, RV^m, \sum D_i; C = 0.6842$	**0.0008178**	0.0016731
SVM Radial Basis	$RV^d, RV^w, RV^m, \sum D_i; C = 3.3, \sigma = 0.005$	0.000819	
K Nearest Neighbor			
KNN	$RV^d, RV^w, RV^m, \sum D_i; k = 29$	0.0009446	0.0017171

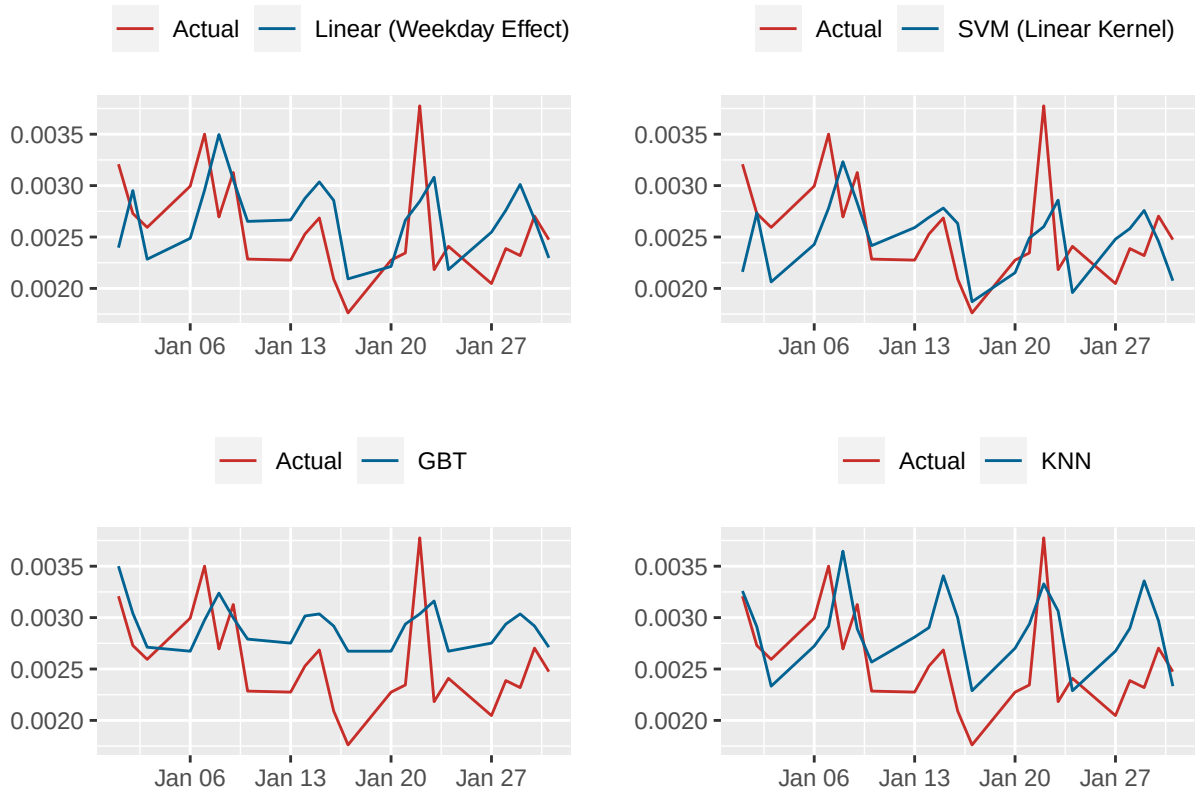
On a first look, of all methods and models considered, the gradient boosted tree algorithm performs best on the out-of-sample `validation` set. Note that on the first pseudo out-of-sample `test` set the support vector machine with linear kernel performed best. I the following, I am analyzing the results in greater detail.



In the plot above, the red line shows the actual volatility, RV_{t+1}, and the other lines are the different models' predictions, $\widehat{RV_{t+1}}$. The plot shows that the first few months of 2020 seem 'stable' and from March onward there is elevated EUR/USD volatility with large differences between actual and predicted values. This is due to market turmoil during the start of the Covid-19 pandemic.

To further investigate how the predictions of the considered models differ, I am comparing the predictions during 'normal' times (January 2020) to periods of market turmoil (March 2020).

8.1 Predictions During Pre-Crisis Period (January 2020)



During 'normal' times;

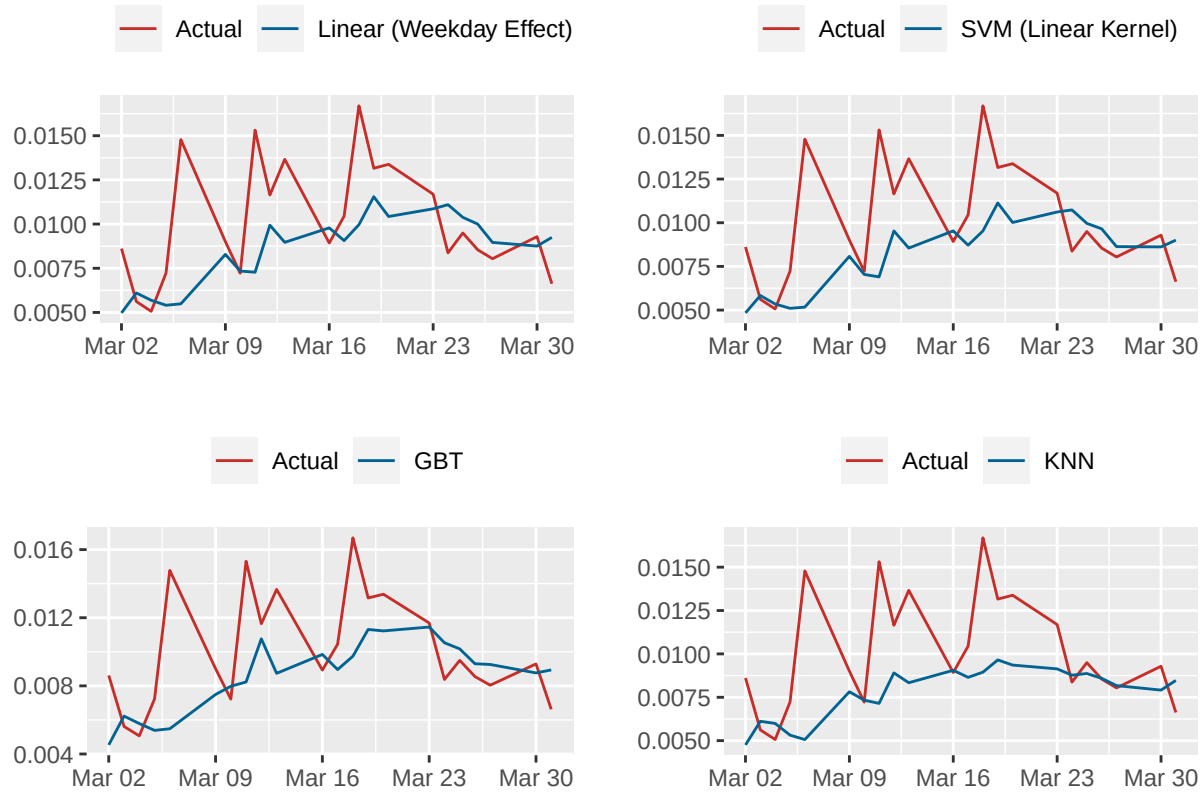
- KNN-based predictions exhibit larger than actual volatility spikes
- Gradient boosted tree (GBT) based predictions are rather flat with *too little* variability in comparison to actual volatility
- The linear weekday effect model and the support vector machine algorithm with linear kernel (SVM) produce volatility spikes that fall in-between the above mentioned two extremes
- The linear weekday effect model appears to *slightly* overestimate volatility, and GBT and KNN *greatly* overestimate volatility
- SVM appears to predict volatility most accurately

For a more objective comparison, below I am computing the RMSE for the month of January 2020.

	$\text{RMSE}_{\text{January 2020}}$	$\hat{P}(\widehat{RV_{t+1}} > RV_{t+1})$	$\hat{P}(\widehat{RV_{t+1}} < RV_{t+1})$
Linear Model (Weekend Effect)	0.0005091	56.5%	43.5%
Gradient Boosted Tree	0.0005415	82.6%	17.4%
SVM (Linear Kernel)	**0.0005008**	56.5%	43.5%
KNN	0.0005499	69.6%	30.4%

As expected by the plots above, the support vector machine (SVM) performs best in terms of RMSE. However, the linear model's performance is only 1.63% worse and this measure is based on merely 23 observations. Both the linear weekend effect model and SVM overestimate volatility in 56.5% of cases. This might be a desirable property in case a 'conservative' estimate for volatility is preferred.

8.2 Predictions During Covid-19 Crisis (March 2020)



During periods of market turmoil (March 2020);

- All models *underestimate* volatility
- Gradient boosted tree (GBT) based predictions have the greatest volatility spikes and seemingly provide the greatest accuracy
- KNN performs worst with rather flat predictions
- The linear weekday effect model and SVM produce very similar predictions, with the linear model's predictions being slightly more accurate as predictions are slightly shifted upwards

	$\text{RMSE}_{\text{March 2020}}$	$\hat{P}(\widehat{RV_{t+1}} > RV_{t+1})$	$\hat{P}(\widehat{RV_{t+1}} < RV_{t+1})$
Linear Model (Weekend Effect)	0.0035152	40.9%	59.1%
Gradient Boosted Tree	**0.003423**	40.9%	59.1%
SVM (Linear Kernel)	0.0036822	36.4%	63.6%
KNN	0.0038273	36.4%	63.6%

The table above confirms what is seen in the plots. Gradient boosted trees perform best in terms of RMSE. Both GBT and the linear weekday effect model *underestimate* volatility in 59.1% of cases, respectively. SVM and KNN perform worse and underestimate volatility to a larger extend.

9 Conclusion

When it comes to predicting financial volatility,

1. During 'normal' times, the support vector machine algorithm with linear kernel outperforms all other considered methods. This is confirmed by two out-of-sample datasets: The `test` set and the `validation` set's *pre-crisis* period
2. During periods of market turmoil, the gradient boosted tree algorithm outperforms all other considered methods. This is confirmed by the complete `validation` set which is heavily impacted by turbulence surrounding the Covid-19 pandemic

The self-developed linear weekday effect model improves over the HAR-RV model by Corsi, F. (2009). Most importantly, the weekday effect model performs *second best* at predicting volatility during normal and crisis times, respectively. In addition, this model is simple to train using least squares estimation. The absence of any tuning parameters and the apparent robustness across market conditions makes this model a well-suited candidate for applications in practice.

As a side note, the general autoregressive model of conditional heteroskedascity (GARCH) by Bollerslev, T. (1987) is shown to perform worst of all models considered. For more detail, please refer to the appendix.

10 Outlook and Potential Improvements

While in the past decades modeling of volatility was primarily driven by parametric models such as the GARCH model (Bollerslev, T. 1987), modern attempts use more accurate measures of volatility as inputs and non-parametric methods for modeling are considered. In the future, researchers might consider modeling volatility using *images* of volatility time-series.

When it comes to the analysis of the feature space given in the dataset for this project, there are several points that are open for future consideration.

1. In this analysis, only 6 different models are considered. However, the `caret` library (Kuhn, M. et al 2020) to date provides 238 algorithms to choose from. It is questionable that the best performing model is included in the investigated models of this paper
2. One could extend the feature space by using *external information*, e.g. CBOE Volatility Index (VIX) quotes, event dates of an economic calendar, EUR/USD option's implied volatility

Furthermore, in this application, the models are fitted *once* on the `training` set and then applied over the *complete period* of the `validation` set. A more rigorous approach would be to only predict *one day* of the hold-out set and then re-estimating the model on a rolling basis of observations. However, this approach is computationally costly, and *each model* considered had to be re-trained (and possibly re-tuned) 153 times to obtain a RMSE measure on the `validation` set.

11 References

Arnold, J. B. et al. 2019. *Package "Ggthemes"*. https://cran.r-project.org/web/packages/ggthemes/ggthemes.pdf.

Auguie, B. 2017. *Package "gridExtra"*. https://cran.r-project.org/web/packages/gridExtra/gridExtra.pdf.

Bollerslev, T. 1987. *Generalized Autoregressive Heteroskedasticity*. Journal of Econometrics.

Breiman, L. et al. 2018. *Package "randomForest"*. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf.

Corsi, F. 2009. *A Simple Approximate Long-Memory Model of Realized Volatility*. Journal of Financial Econometrics.

Dowle, M. et al. 2020. *Package "Data.table"*. https://cran.r-project.org/web/packages/data.table/data.table.pdf.

He, T. et al. 2020. *Extreme Gradient Boosting*. https://github.com/dmlc/xgboost.

Irizarry, Rafael A. 2020. *Introduction to Data Science*. https://rafalab.github.io/dsbook/.

Kaplan and Schlegel. 2020. *Package "fastDummies"*. https://cran.r-project.org/web/packages/fastDummies/fastDummies.pdf.

Karatzoglou, A. et al. 2019. *Package "Kernlab"*. https://cran.r-project.org/web/packages/kernlab/kernlab.pdf.

Kuhn, M. et al. 2020. *Package "Caret"*. https://cran.r-project.org/web/packages/caret/caret.pdf.

R Core Team. 2020. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org.

Sharelab. 2020. A platform for US equity analytics: SHARELAB LIMITED. https://www.sharelab.com.

Spinu, V. 2020. *Package "Lubridate"*. https://cran.r-project.org/web/packages/lubridate/lubridate.pdf.

Whitley and Ball. 2002. *Statistics Review 6: Nonparametric Methods*. Critical Care.

Wickham, H. 2019. *Package "Tidyverse"*. https://cran.r-project.org/web/packages/tidyverse/tidyverse.pdf.

Wuertz, D. et al. 2020. *Package "fGarch"*. https://cran.r-project.org/web/packages/fGarch/fGarch.pdf.

Xie, Y. et al. 2020. *Package "Knitr"*. https://cran.r-project.org/web/packages/knitr/knitr.pdf.

12 Appendix 1: Financial Volatility

In this section I am providing a brief exursion for those unfamiliar with financial volatility.

Financial market participants trade assets such as stocks, currencies, bonds and commodities. The prices of these assets are continuously changing. These price fluctuations are measured in terms of 'volatility'.

Mathematically, volatility refers to the *standard deviation* of the asset's *price changes*. The greater the volatility of an asset, the more *uncertainty* there is regarding the asset's price developments.

Example

Consider two assets, A and B. In the future, asset A yields a return of \$100. For asset B, the return is going to be either \$200 or \$0 with equal probability. Note that asset A yields the same *average* return as asset B, i.e. \$100. However, for asset A there is *no risk* of ending up with nothing. If both assets were priced equally, the *risk-averse* investor would always choose asset A over asset B, as it has the same *average payoff* but without any risk. In this example, asset A has a volatility of 0, while asset B has a volatility of 100:

$$\sigma_B = \sqrt{\sum_{i=1}^{N} \mathrm{p}_i (x_i - \mu)^2} = \sqrt{0.5(200-100)^2 + 0.5(0-100)^2} = 100$$

The example demonstrates how greater uncertainty regarding an asset's price developments is associated with greater volatility. The greater the volatility of an asset, the more risk is borne by the investor. In theory, investors demand a 'risk premium' (a greater average return) to assume this risk. Popular risk measures such as value at risk (VaR) and expected shortfall (ES) are often modeled using an asset's volatility.

12.1 Conditional Heteroskedascity

Financial volatility is not constant over time. The standard deviation of asset price changes varies and exhibits significant time-dependencies. In practice, this means that large *price changes* of either sign are more likely to be followed by large price changes of either sign. Likewise, small price changes are likely to be followed by small price changes. Hence, volatility is *conditional* on past volatility.

$$E[\sigma_{t+1} | \sigma_t, \sigma_{t-1}, ..., \sigma_{t-n}]$$

In this context, the *time-dependency* of the standard deviation of price changes (σ_t) is referred to as *conditional heteroskedascity*.

In academic literature, the GARCH model (Bollerslev, 1986) is widely used to capture the conditional heteroskedascity of financial time series. The standard GARCH model has a similar structure as the autoregressive moving average (ARMA) model. However, while an ARMA model captures the conditional mean of a process, the GARCH model captures the conditional variance of a process.

When it comes to the dataset provided, we are presented with a problem to forecast volatility given different aggregations of past volatility. This calls for a supervised learning approach of modeling volatility as described in this report. In the appendix, I am merging external EUR/USD return time-series to the given dataset and compare the results of this project to the performance of a GARCH(1,1) model.

13 Appendix 2: GARCH(1,1) Volatility Forecast

Using *external information*, it is possible to merge daily EUR/USD quotes, derive the continuously compounded, squared return, fit a GARCH(1,1) model on the `training` set and compare the GARCH(1,1) RMSE performance to the other algorithms on the `validation` set.

Considering the application at hand under the GARCH(1,1) model, the conditional variance σ_t^2 is dependent on the continuously compounded past EUR/USD return r_{t-1} and the past value of σ_t^2, i.e. σ_{t-1}^2:

$$\sigma_t^2 = \omega + \alpha r_{t-1}^2 + \beta \sigma_{t-1}^2$$

where:

t: Time index (here: consecutive trading days of the EUR/USD currency pair)

σ_t^2: Conditional variance

ω: Intercept parameter

α: Parameter value for r_{t-1}^2-variable

β: Parameter value for the lagged conditional variance σ_{t-1}^2

r_{t-1}^2: Continuously compounded and squared asset return of the last observed trading day, $t-1$

Note that for the GARCH(1,1) model the input variables are r_t^2 and $\hat{\sigma}_t^2$, whereas the other methods considered have a different feature space as described in the variables and data exploration sections.

In this application, we are interested in 'volatility', i.e. the conditional standard deviation σ_t. Also, with data given on date t we are trying to predict volatility for the next trading date, i.e. $t+1$. Hence, the time index is shifted by one day:

$$\sigma_{t+1} = \sqrt{\omega + \alpha r_t^2 + \beta \sigma_t^2}$$

The `garchFit` function of the fGarch' library conveniently estimates the parameter values $\hat{\omega}$, $\hat{\alpha}$ and $\hat{\beta}$ using maximum log-likelihood estimation.

Analog to the previous procedures, I am fitting the GARCH(1,1) model on the `training` set and investigate the prediction performance on the `validation` set.

Estimated Parameter Values

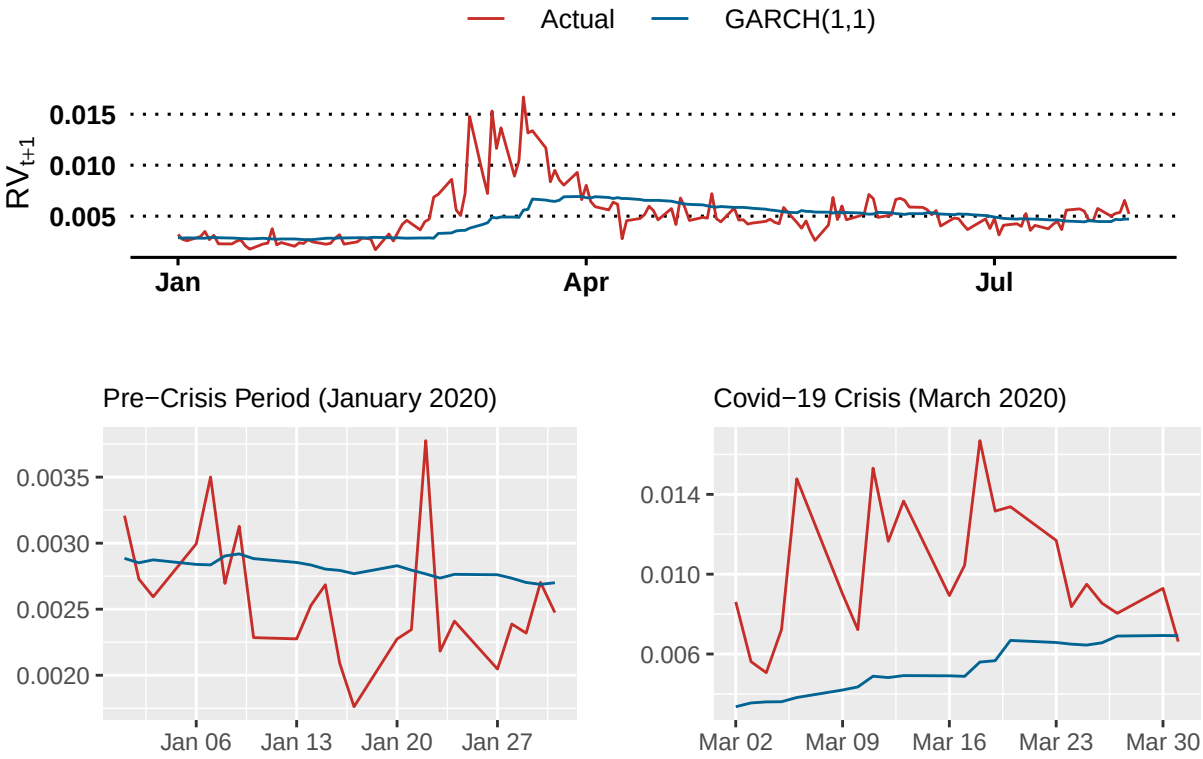
$\hat{\omega}$	$\hat{\alpha}$	$\hat{\beta}$
1e-07	0.0306548	0.967251

13.1 GARCH(1,1) Out-of-Sample Performance

Model	Variables	$\text{RMSE}_{\text{test}}$	$\text{RMSE}_{\text{validation}}$
Benchmark	μ	0.0024906	
Linear Model			
HAR-RV	α, RV^d, RV^w, RV^m	0.0008829	
Weekday Effect	$\alpha, RV^d, RV^w, RV^m, \sum D_i$	0.0008568	0.0016219
Tree Ensembles			
Random Forest	$RV^d, RV^w, RV^m, \sum D_i$; mtry $= 2$	0.0009372	
Gradient Boosted Trees	$RV^d, RV^w, RV^m, \sum D_i; \alpha = \lambda = 0.019$	0.0008966	**0.0015951**
Support Vector Machine			
SVM Linear Kernel	$RV^d, RV^w, RV^m, \sum D_i; C = 0.6842$	**0.0008178**	0.0016731
SVM Radial Basis	$RV^d, RV^w, RV^m, \sum D_i; C = 3.3, \sigma = 0.005$	0.000819	
K Nearest Neighbor			
KNN	$RV^d, RV^w, RV^m, \sum D_i; k = 29$	0.0009446	0.0017171
Appendix: GARCH(1,1) Model			
GARCH(1,1)	r_t^2, σ_t^2		0.0024647

As seen in the plot above, the GARCH(1,1) model fails to achieve the performanc of all other methods considered.

Below, I am visualizing the *out-of-sample* `validation` predictions $\hat{\sigma}_{t+1}$ vs. the actual (realized) volatility RV_{t+1}.



The plots show that;

- The GARCH(1,1) model's predictions are rather flat and fail to match the actual volatility spikes by a great margin
- During 'normal' times, the GARCH(1,1) model, on average, seems to neither under- nor overestimating volatility
- During times of market turmoil, the GARCH(1,1) model greatly underestimates volatility

In conclusion, the GARCH(1,1) model based on squared asset returns is outperformed by all models considered in this paper (excluding the constant benchmark 'mean' model).