Predictive Models on Tesla Stock

Samuel Sekarski
Supervisor: Dr. Gholam

June 2021

Contents

List of Figures

Chapter 1

Introduction

In 2020, despite the global pandemic, the electric vehicle maker Tesla saw it's stock price go up 700%, which is more than Amazon's or Netflix's went up, both of whom profited during the pandemic. Tesla was also added to the S&P500 index in November and underwent a stock split shortly before that, because of increased demands. In fact for years, Tesla stock has been on the minds of more and more potential investors, wondering if they should buy-in or not.

In this thesis, we will look to try to find models that could accurately fit Tesla's stock price evolution, which people could use to help make decisions. We will not, however, develop any kind of trading strategy, nor suggest to the reader any financial actions.

To frame our research our main question will be: How well can existing models fit Tesla stock price ? We shall use data spanning 2019 and 2020.

The classical approach to financial data is through Time Series (TS) analysis, and as such we shall consider various models developed for time series. The more novel approach is to use Machine Learning (ML) algorithms to attempt to train a model to fit the financial data. We shall explore some of these methods as well, and then compare both the Machine Learning and Time Series approaches, and see which yields the better fits. This shall be our second area of focus.

In continuation of the first question, we shall ask ourselves how public sentiment about Tesla affects it's stock. In the past, investing was reserved only to big institutions and select individuals, but with the rise of internet, this practice has been expanded and now anyone (so called "retail investors") can invest. And as seen in the Gamestop short squeeze of January 2021, when a group of individuals have a similar sentiment, they can influence the markets immensely. We shall use Google Trends to see if adding a public sentiment factor into account benefits the modelling.

Another subquestion we shall consider, is whether posts on social media can influence a stock. Tesla CEO Elon Musk is known for his tweeting habits that can shake markets, as for example with his tweets about cryptocurrency in early 2021. Another example, closer to the subject, is when Musk tweeted in May 2020 that the stock price was "too high" and the stock then fell 10%. The US

Securities and Exchange Commission (SEC) definitely thinks Musk's tweet have influence, as the SEC filed a lawsuit against the billionaire in September 2018 regarding one of his tweets [1]. We shall use data collected from Twitters API to see if his tweets truly have an influence and if using this data we can improve upon our models.

We shall also, in a lesser measure, compare how implementations of these methods in the two main programming languages for data analysis, that is Python and R, compare to each other. Are there significant outcome differences using one or the other ? What kind of support (frameworks, libraries, packages, etc.) exist for this kind of modelling in each ?

And finally, since the beginning of 2021, the stock price has tapered off, and fallen. We shall lastly ask ourselves if our best models developed on the 2019 and 2020 time frame will predict this.

[1] `https://www.sec.gov/news/press-release/2018-226`

Chapter 2

Tools

Python will be our main tool for analysis, but we will also consider using R, and compare, where appropriate, the results using Python versus using R, to see if there are any differences in implementation that influence the outcome of our analysis.

2.1 Python

Python is a multi-purpose high level programming language, which is gaining a lot of traction in the machine learning community, for its simple syntax and library support. The main package we will be using for Machine Learning is 'sklearn' (`https://scikit-learn.org/`). The main pacakge we will be using for Time Series is 'statsmodels' (`https://www.statsmodels.org/`). For GARCH we will use the 'arch' package (`https://arch.readthedocs.io/`). For data manipulation we will be using 'pandas' (`https://pandas.pydata.org/` and 'numpy' (`https://numpy.org/`)

2.2 R

R is a programming language designed specifically for statistical computing and graphics and has been the go-to programming language for data scientist in the past. It has a lot of library support as well.

The main package used is 'tidyverse' (`https://www.tidyverse.org/`). This is actually a collection of packages, that all share the same underlying structure, and that are data science orientated.

For GARCH modeling we will be using the 'fGarch' package (`https://cran.r-project.org/web/packages/fGarch/index.html`)

Chapter 3

Data

Realizing that we need to be careful with time zones, as for example, TSLA is traded on the NASDAQ stock exchange, which is in New York, whereas Elon Musk tweets from California and I am in Switzerland while analysing the data and the reader might be anywhere in the world, we shall convert all times to UTC.

3.1 Main data: Tesla Stock Price

The TSLA price data was retrieved using the python package "yfiance" which makes calls to the Yahoo Finance API. We were able to download 730 days worth of hourly data, spanning the last 730 days from download date. The Yahoo Finance API is not being maintained, so this may not be possible in the future. The data was retrieved on February 10th 2021 and spanned from February 11th 2019 at 00:00 till February 10th 2021 at 00:00. The raw data contained the following variables: "Open", "High", "Low", "Close", "Volume", "Dividends" and "Stock Splits". For the purposes of this thesis, we only kept "Open" which was an arbitrary choice.

3.2 Auxiliary data

3.2.1 Google Trends

Google Trends (GT) is a tool that lets researchers explore how the Google search engine as been used. We shall use the search term "tesla" in the explanation and graphics, as the other search terms ('tsla', 'tesla stock' and 'musk') were similarly preprocessed.

The Google Trends data was retrieved using the python package "pytrends" which makes calls to the Google Trends API. Each API call will return the percentage of the number of searches per time unit in an interval, relative to the peak searches of the interval. That is to say that at the time step where

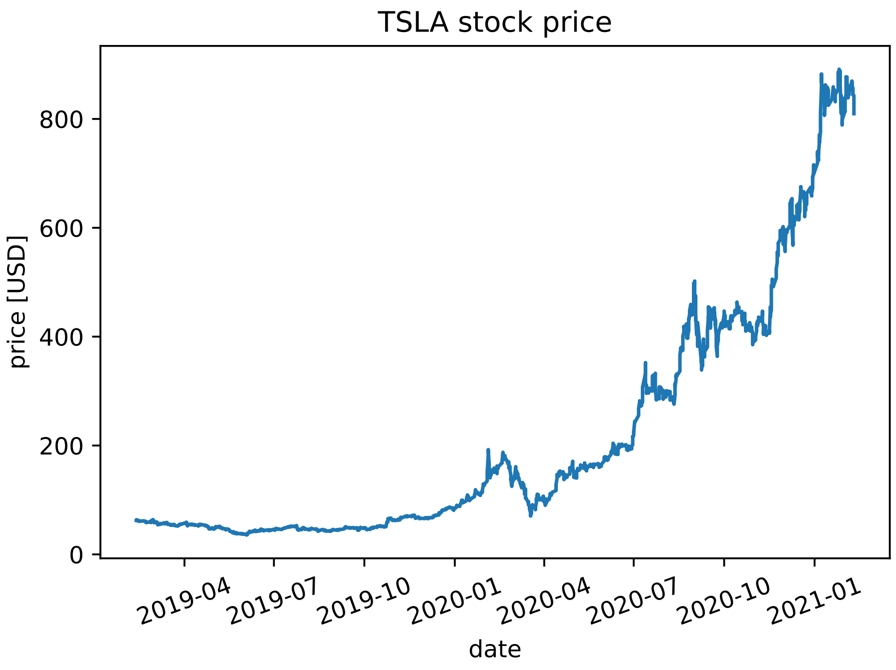

Figure 3.1: TSLA hourly opening stock price

there were the most searches, the returned value will be 100, and for all other time steps, the return value will range from 0 to 100, rounded to the nearest digit. We are interested in hourly data, but for hourly granularity, we can only retrieve one week at a time. Retrieving a longer interval results in a higher granularity. The solution we used was to do multiple calls to the API to get the hourly data relative to each week, and then also retrieve the weekly-granular data for the entire timeframe, and compose the hourly with the weekly, to get hourly percentage relative to the whole time interval.

The data was retrieved on March 17th 2021 and spanned from December 31st 2018 at 06:00 to March 17th 2021 at 15:00. After visual and numerical inspection of the data (fig 3.2, it appeared that there was a gap in the data between February 24th 2020 12:00 and March 17th 2020 15:00. I can only hypothesis that there was some Covid-19 induced problem at Google that is responsible for this, as that's the time frame where the Pandemic started to really pick up globally, as the same gap is present in all searches that I looked up with their API. We truncated this missing data, with the daily GT value, which were available. We did a simple step truncation, before composing the hourly-weekly data with the weekly-interval data. As shown in Fig 3.3

We observe in this figure also, that there seems to be some daily periodicity in the searches, which is explained by the fact that we only looked at searches

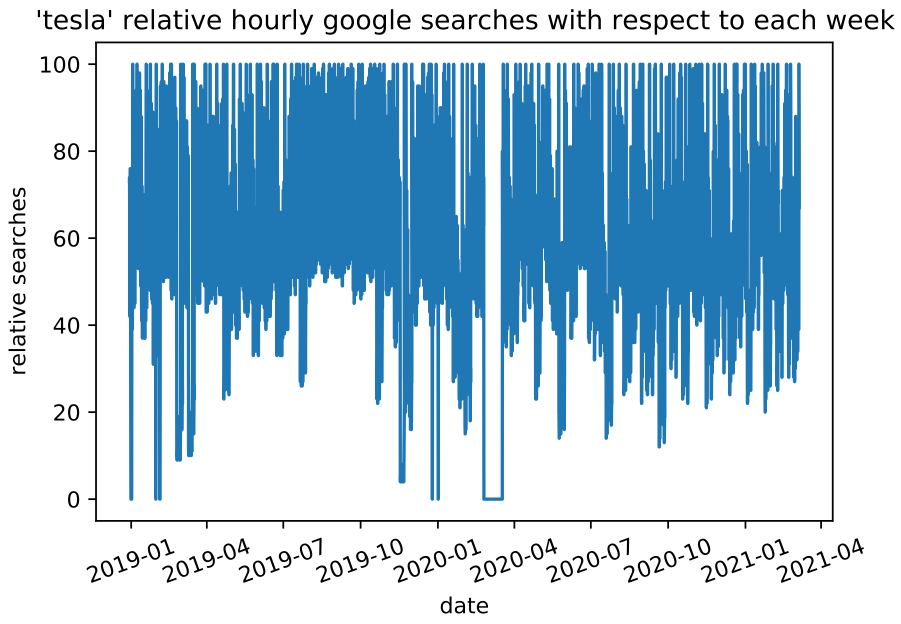

Figure 3.2: Hourly google searches of the term ”tesla” relatively to each week, in percentage of the weekly maximum

done in the United States, so during night time hours it makes sense that there would be less searches.

As the trading data is only between 10am and 5pm EST, we simply dropped data outside of that fork. This was a subjective choice, and it can be discussed whether it would not have been better to aggregate the dropped time and incorporate them in the observations somehow. Our reasoning was that if people searched for the terms before the market opening (say at 11pm the previous day), because they were interested in buying, they were likely to search for the term again right before the market opens, in order to make sure no new information came to light in the mean time to change their mind, and thus their interest would still be captured in the used data. Likewise, if someone was searching with the intent on selling, they too would search again prior to making the transaction, in order to double check.

The final preprocessed data can be seen in figure 3.4

3.2.2 Twitter data

The popular social media platform Twitter has an API that lets researchers interact with and explore the Twitter Database. There are several tiers available that give different degrees of access to the database. The free tier did not give significant access to the tweet history, and as such we had to apply for an

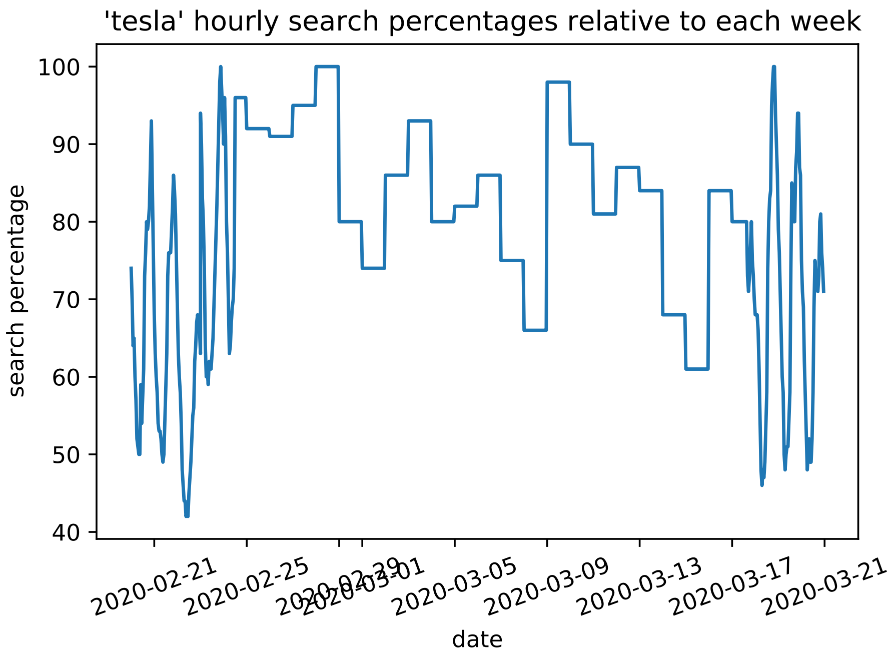

Figure 3.3: Google searches for ”tesla” between February 20th and March 20th 2020, where the missing hourly data as truncated with daily data

academic tier.

The API gives access to an impressive amount of data. We will be focusing on tweets by Elon Musk, and twitter metrics of ”like count” and ”retweet count”, though others are available as well. These are the most used metrics when judging a tweets popularity. Even if they both measure a tweets popularity, if they are not very correlated it could be worth keeping both of them, as a like only shows approval of the tweet, whereas a retweet shows approval, but also spreads the tweet to a larger audience. So it could be used as a metric of how much a tweet has spread.However, upon inspection (Fig 3.5) the retweet count and the like count seem pretty correlated, so we will focus only on the like count which will be both a measure of the approval of the tweet as well as a measure of how viral it is.

Tweets have a precise timestamp and thus we have aggregated them to that for every hour H, the corresponding value is that of the sum of all tweets in the range]H-1,H]. We then added the out of trading hours values to the first trading hour of each day. Again, it can be discussed whether it would have been better to aggregate this data differently, or drop it all together, like for the Google Trends data. Our reasoning is that if Elon Musk tweets something out of hours, that gets people in a good or bad frenzy about TSLA, then that frenzy would not die down before the market opened.

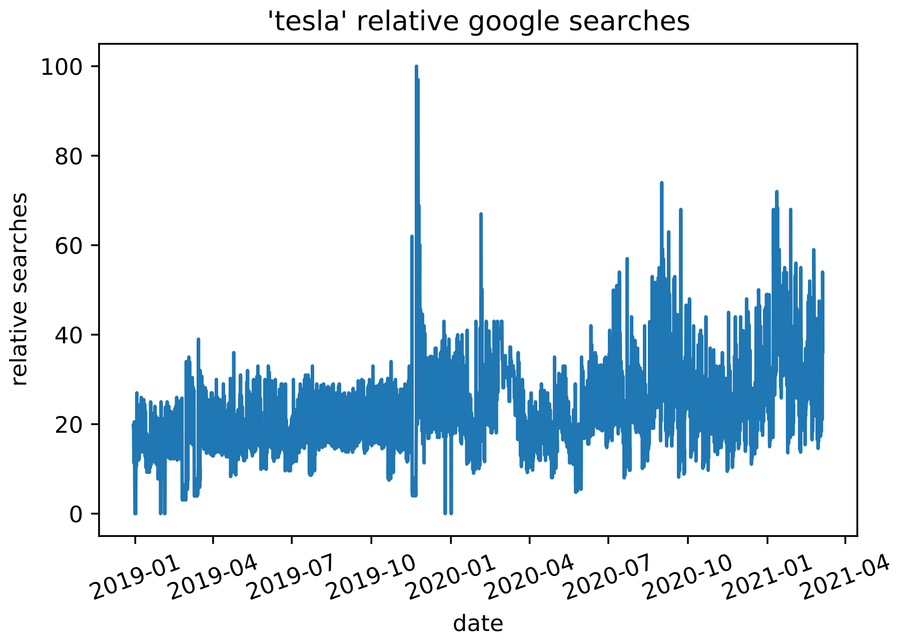

Figure 3.4: ”tesla” keyword relative google searches

Elon Musk tweets a lot about things unrelated to Tesla and it’s stock as well, so we will filter tweets with the term ”Tesla”. This may be too much filtering, as it could be imagined that other keep words could be worth adding to the filter, but it should be decent enough.

The final aggregated like count per period can be seen on figure 3.6

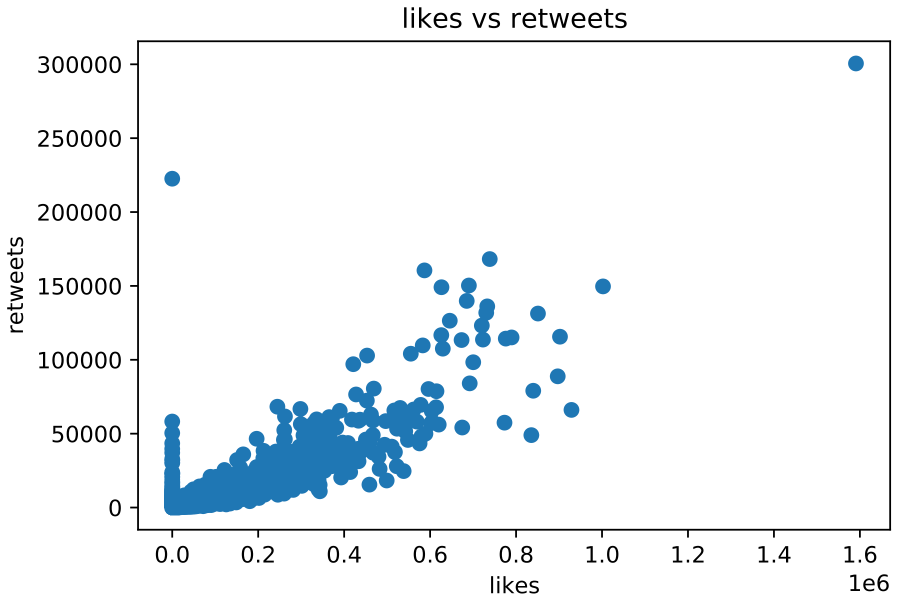

Figure 3.5: Elon Musk's tweets like count vs retweet count

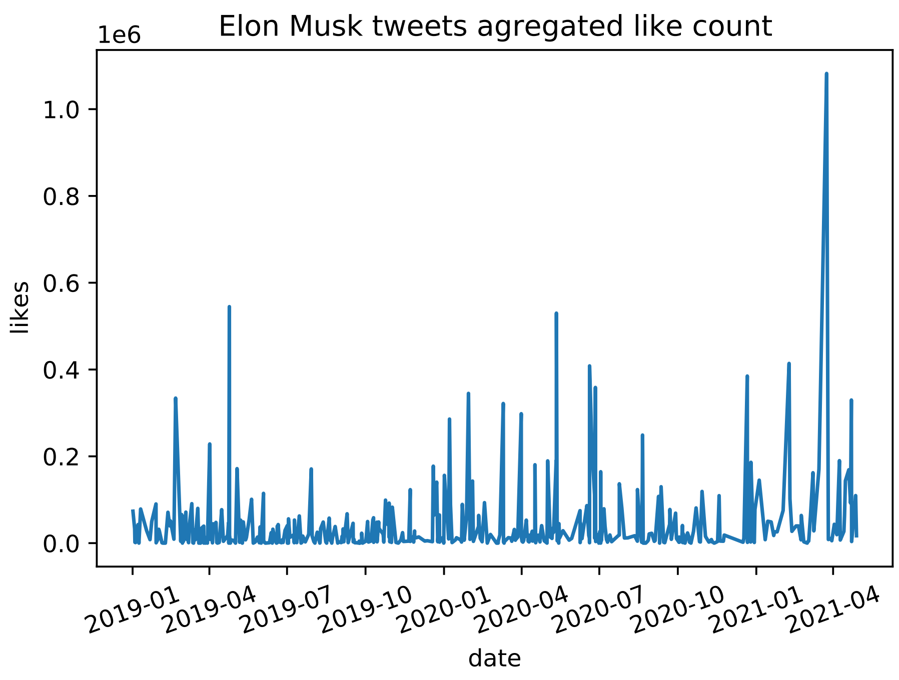

Figure 3.6: Elon Musk's tweets like count, aggregated to fit the trading hours of NASDAQ

Chapter 4

Experiment Design

Let $(\mathcal{X}, \mathcal{Y})$ denote our data, where $\mathcal{X}$ is the input data (we will use the terms "features", "predictors" and "variables" as well throughout this paper) and $\mathcal{Y}$ is the output data (we will also use "response"). We shall separate all our data into 2 sets: the training set $(\mathcal{X}_{train}, \mathcal{Y}_{train})$ and the test set $(\mathcal{X}_{test}, \mathcal{Y}_{test})$ The split date was chosen to be January 4th 2021 at 10:00. This was a rather arbitrary choice, to leave about one months worth of data for testing. Let us denote the split date as T We will train our models on the training set, and then predict responses for observations in the test set. That is we will predict a set $\mathcal{Y}_{predict} = F(\mathcal{X}_{test})$ where $F()$ is the model's prediction function. As the stock price will be used both as features and as responses, we shall denote it as follows: Let P_{T-t} denote the stock price at timestep t with respect to the split.

4.1 Metrics

To compare the results and rank them, we shall use several metrics to asses goodness of fit and predictive power. The main two we shall consider are Mean Square Error (MSE) and R^2.

4.1.1 MSE

Let m be the size of the test set and let $\hat{Y}_i$ be the predicted value for the i-th observation in the test set. The Mean Square Error is defined as $MSE(\mathcal{Y}_{predict}) = \frac{1}{m}\sum_{i=1}^{m}(Y_i - \hat{Y}_i)^2$

4.1.2 R^2

R^2 denotes the coefficient of determination.

The one used by our Python package, is defined as:

$R^2(\mathcal{Y}_{test}, \mathcal{Y}_{predict}) = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y}_i)^2}$

Where $\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$.

The best score is 1.0, a constant model that always predicts the expected value of y without taking into account the input, would be 0.0 and the score can be arbitrarily negative, as the top term can be arbitrarily large (i.e. our predictions can be arbitrarily bad. So for our methods to be better than just using the mean of the observations, we would like them to be larger than zero.

4.2 set-up

4.2.1 Times Series

For the Time Series set-up, the response variable is $\mathcal{Y} = \{P_t\}$ and the features are $\mathcal{X} = \{(P_{t-1}, \ldots, P_{t-p})\}$ and where p is a parameter that needs to be determined.

4.2.2 Univariate response

In this set-up we consider the response variable to be univariate $\mathcal{Y} = \{P_{T+t}\}$ and features to be $\mathcal{X} = \{(t, GT1_{T+t}, GT2_{T+t}, TW1_{T+t}, TW2_{T+t})\}$. Where the variables are from left to right: time, the google trend for 'tesla', the google trend for 'musk', Elon Musks tweet likes, Elon Musks tweet retweets, all at time $T+t$

We shall then train the models on the training set, and then predict one week ahead, and compare with the values of the test set.

4.2.3 Multivariate response

In this set-up we consider the response variable to be multivariate $\mathcal{Y} = \{(P_{T+t}, \ldots, P_{T+t+r})\}$ We shall consider the response to be one week ahead worth of data, or $r = 35$ steps. To predict this multivariate response, we shall augment the feature space dimensions, by not only considering X_{j_T}'s as predictors, but also $X_{j_{T-1}}, X_{j_{T-2}}, ..., X_{j_{T-p}}$ as predictors (Where the X_j are the same variables from the univariate case). We shall consider one month of past data (or $p = 140$ time steps) as part of one observation. A sort of hybrid Time Series - Machine Learning approach.

In the first set-up, the forecast of observation Y_{T+1} is generally good but the forecast of each observation after that gets more and more vague, faster and faster. To the point where after a number of time steps, the forecasts hold no more value and become effectively useless. The idea in the second setup, is to put a certain number of future observations into the response variable, so that when forecasting, the "first observation" that is forecast (so the one that is usually the best) now contains contains multiple steps ahead. Of course this comes at a price; using the same feature space, but with more response variables, will cause the fit and predictions to be worse. So we need to augment the feature space as well, by considering multiple steps behind as part of one observation point. This is actually what the Time Series methods actually already do. They use a combination of previous values to predict the next values.

We shall then train this on the training set, and predict one step ahead (which effectively will be one week ahead) and compare the values with those of the test set.

Chapter 5

Methods

5.1 Time Series Methods

We'll go over the various methods we will be using, giving a brief description and explaining in what cases they are used. Unlike with regular regression, Time Series analysis generally doesn't use auxiliary data to predict how the series will behave, but exploits intrinsic relation within the series itself. Often using some combination of previous values and randomness to predict the next value. The data must therefore have a Time Series representation. That is, we can index it by an ordered set, such as time. Let us denote such a series by $\{Y_t\}_{t=1,\dots,n}$. It is generally assumed that a Time Series has to be stationary to work with. In other words, the series should have a constant mean, and a covariance function depending only on the lag between 2 observations (and not for example of the time step). In practice trend is removed from the series before analysis all together and then is added back at the end if necessary. Classic Time Series methods are generally good at explaining TS, but are less interesting for predictions, as they always predict the expected value, as we will see with our Tesla case.

5.1.1 ARMA

Autoregressive Moving Average (ARMA) model have 2 components: an Autoregressive (AR) part and a Moving Average (MA) part. The AR part, as it's name suggests, is a regression of the series with itself. That is, we suppose that an observation is a linear combination of a certain number of previous observations, plus a little noise. Here the noise of each observation has an indirect influence on following values. The MA part, on the other hand, assumes that the error terms of previous observations have a direct influence on following values. In other words, we use previous error terms as covariates to do regression on the series. Putting both together we get the general ARMA model. An ARMA (p,q) model is defined by:

$$Y_t = \sum_{j=1}^{p} \phi_j Y_{t-j} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t$$

Where the ϕ_j's and θ_j's are constants with $\phi_p, \theta_q \neq 0$. Usually $\epsilon_t \overset{iid}{\sim} \mathcal{N}(0,1)$ but could be any distribution with zero mean and unit variance.

5.1.2 GARCH

ARMA models make an assumption that can be quite limiting for financial Time Series, in that is assumes the variance is homogeneous throughout time, but with financial data this is practically never the case. There are always periods of more volatility than other, for example when there is a big announcement or a scandal, or other speculative uncertainty. To address this problem, a model was developed for modeling noise in a Time Series that was variable: the Autoregressive Conditional Heteroskedasticity (ARCH) model, and later it's generalization, the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model.

The idea of these models, is to model the variance as an ARMA process. This leads to the following definition.

A GARCH(m,r) model is defined by:

$$Y_t = \sigma_t \epsilon_t$$

$$\sigma_t^2 = \alpha_0 + \sum_{j=1}^{m} \alpha_j Y_{t-j}^2 + \sum_{j=1}^{r} \beta_j \sigma_{t-j}^2 \quad , \epsilon_t \overset{iid}{\sim} F(0,1)$$

where F is any distribution with zero mean and unit variance. Often it is either Normal or Student t (that has been standardized). GARCH models are generally the Time Series models used for financial data, but more generally it is used any time the variance is clearly not constant. A GARCH model can also be combined with an ARMA model, to create and ARMA-GARCH model. As we will see later on, this will not be necessary here so we will not go further into these models.

5.2 Machine Learning Methods

We'll go over the various methods we will be using, giving a brief description and overview of the pros and cons of each and when it is appropriate to use them. Following the same notation as in chapter **??**, let $(\mathcal{X}, \mathcal{Y})$ denote the features and responses of our data. A machine learning model is some function that takes the features $\mathcal{X}$ as inputs and returns an approximation of the responses $\hat{\mathcal{Y}}$ as output. Let us denote a generic model by F in such way that $\mathcal{Y} \approx \hat{\mathcal{Y}} = F(\mathcal{X})$. The goal of ML is to find methods that give good approximating functions F.

The two questions that arise are "What do we use as approximating function F ?" and "What makes and approximation 'good' ?" We will tackle the second question later. In the following we shall explore various methods (models F) that have been developed.

First, however, as we are dealing with finite sets, we will suppose that there are n observations and m features. In this section we will only specify the methods for the univariate response case, and not go into details about the generalization to multivariate responses. This all allows us to rewrite the above sets as:

$$\mathcal{X} = \{(X_{i1}, \dots, X_{im}) : i = 0, \dots, n\}$$
$$\mathcal{Y} = \{Y_i : i = 0, \dots, n\}$$
$$\hat{\mathcal{Y}} = \{\hat{Y}_i : i = 0, \dots, n\}$$

And the model relation as:

$$Y_i \approx \hat{Y}_i = F(X_{i1}, \dots, X_{im}) \forall i \in \{1 : n\}$$

For simplicity in notation, we will often use $\vec{X}_i = (X_{i1}, \dots, X_{im}$.

We are now ready to explore the first method.

5.2.1 Linear Models

The idea is to use a linear combination of features to determine the response. The linear constants are the parameters that need to be chosen to give the best approximation of the response space using the feature space. Geometrically this means fitting a hyperplane to the training set, by minimizing some function (loss function). Formally:

$$\hat{Y}_i = \sum_{j=1}^{m} \beta_j X_{ij} \quad , i = 1, \dots, n$$

Advantages

- Very well understood

Linear Model are some of the most common and simple models out there, but they have proven to be reliable and can be made quite complex with feature engineering. There plenty of theory and proofs about these methods. This all makes these methods and obvious first choice of methods to consider.

Problems

- Linearity assumption

Assumes that the outputs are a linear combination of quantities. Although the fact that the linearity is in the parameters leaving a lot of room for complexity with feature engineering (like for example adding an artificial feature that is the square of another feature), this is still limiting as not all relations between the data can be expressed.

- Suffers from multicolinearity

In it's basic form, Linear Model uses Ordinary Least Squares (OLS) which assumes the features are independent. If some are highly colinear, the method becomes unstable and small small random errors in the observations will produce a large impact, thus increasing the variance. This problem can be addresses partially by regularization (or shrinkage): putting a penalty on the size of the parameters. Example of this are Ridge and Lasso regression.

5.2.2 K Nearest Neighbours

The intuition behind KNN methods is simple. The idea is that observations (points in the feature space) which are "close" will have responses that are also "close". Then a good approximation would be given by averaging the responses of the K observations "closest" to the new observation. In this model, we need to chose the definition of "close" as well as chose the value of K.

Formally:

$$F(\vec{X_+}) = 1/K \sum_{i:\vec{X_i} \in N(X_+)}^{Y_i}$$

Where the neighborhood $N(\vec{X_+})$ of $\vec{X_+}$ is the collection of K closest observations

Usually, the distance used to define "close", is the euclidean distance and that is the one we used. As for K this parameter needs to be tuned. The larger it is the smoother the result is and the lower the more jagged it is. For optimal tuning the parameter should be as small as possible, without over-fitting.

Advantages

- Intuitive

The concept is very simple to understand, we are just taking the average of a new point neighbors to determine the output

- Simple to implement

It takes only a few lines of code to implement KNN and get it up and running.

- Non parametric

No prior knowledge about how the data is actually distributed is made, which makes it a flexible tool that can be used anywhere.

Problems

- Curse of dimensionality

It suffers from curse of dimensionality. When the dimension of the feature space gets large, the space becomes sparsely populated, and the K closest observations, might actually be very far from each other, and thus not reflect well the new observation. As a workaround for this problem, there is a variant of KNN which instead of using the K closest observations, uses all the observations which fall in a certain radius of the new observation point. Thus the K parameter becomes R, the radius of the hyperball. A problem with this version is that there may be no observations that fall in the hyperball of a new data point and then nothing can be inferred about it.

- Piecewise constant

The response space is discretized so not all outcomes can be predicted.

5.2.3 Trees

To understand more sophisticated tree methods, we first need to understand how a simple tree works for regression. In KNN, the feature space in separated into regions (or "strats") and fits a constant to each region. Decision Trees work much the same way, but allow for different and more complex definitions of strats. A simple binary tree, or a Decision Tree, work by splitting the feature space recursively into simple strats. Each leaf of the tree corresponds to a strat and the constant is the average of all observations that correspond to that leaf. A fully grown tree, i.e. where each observation corresponds to one leaf exactly, is equivalent to 1-NN Once a tree has been grown, the predictor $\hat{Y}_i$ of $\vec{X}_i$ is the average of all the Y_i in the strat that $\vec{X}_i$ falls into. The prediction function is therefore piecewise constant. Formally:

$$F(\vec{X}_i) = \sum_{l=1}^{L} c_l I(\vec{X}_i \in A_l)$$

Where there are L strats denoted by A_l and c_l are the averaging constants. It is NP-hard to determine the optimal Decision Tree for a given data set, and thus greedy approaches are used. Each split is determined by the variable and threshold that minimize the Mean Square Error of the split.

Advantages

- Non parametric

Trees make no assumption on how the data is actually distributed and can be used on any data. Very little thought has to go into it

- Logarithmic complexity

Trees are inexpensive to use to predict values as they have a logarithmic complexity due to the binary splitting nature of Trees.

- Easy to understand

The concept of Decision Trees are very simple and they can be visualized, which help to understand them.

Disadvantages

- High variance

The choice of each split, especial the first ones, can greatly influence the outcome of the model and thus Decision Trees have high variance.

- Over-fitting is easy

If we grow the Tree out completely, we will end up with a model where every leaf explains exactly one observation. This is a case of extreme overfitting, but even if we stop earlier we can easily over fit if the stopping criteria is not well decided.

- Unstable

A tree could look completely different if the first split is different or any subsequent split is different (but with decreasing impact). This means Trees are unstable; a slight change in the data might completely change the outcome.

- No optimality

In theory there exists an optimal Decision Tree, but the problem of finding the optimal Tree is NP-hard, so in practice we use a greedy algorithm to grow the Tree. This means the final Tree might not be close to the optimal tree at all.

- Not smooth

The model is piecewise constant, and so the prediction space is not smooth.

The variations of Decision Trees we shall consider are

Random Forest (RF) work by taking many Decision Trees constructed by taking random samples of the training set, with replacement, and then averaging the trees. At each split a random number of variables are chosen, thus ensuring that if a variable is predominant, it won't always be chosen first in each tree of the forest, thus decreasing the correlation of the trees amongst themselves and in turn increasing the reduction in variance. This will reduce the variance, with a cost of raising the bias a little.

Extremely Randomized Trees (ET) are random forests, but in each tree, the split is determined by a random threshold, decreasing even more the correlation of trees. Again this leads to less variance but a slightly higher bias

5.2.4 Artificial Neural Networks

A fully in-depth and formal explanation of Neural Networks would take us out of the scope of this report. The literature on the subject is immense and the idea dates back to the 1950's and has grown ever since. As such we will give only a brief and simplified explanation in this report. The idea behind Artificial Neural Network (ANN) is to try to imitate the way neurons work in a brain: the neurons are connected together in a complex network, and when they are stimulated by inputs enough to reach a certain activation level, they will transmit a signal to certain neighboring neurons, combining signals, and so forth until some output is produced.

An ANN consists of an input layer, and output layer and multiple hidden layers (see Figure 5.1). The inputs are linearly combined, if activated, into the next layer of neurons, which is combined again, and again until reaching the output layer.

Advantages

- Can learn non-linear models

The models can be use on a wide variety of problems, including those which have non-linear properties, and if tuned right work relatively well. A lot of people don't even think too much about the regression problem they have and just throw a Neural Network at it.

Problems

- Non-convex loss function

This leads there to be multiple local minimum, and a different random weight initialization can lead to very different results

- Hyperparameters tuning

There are a lot of parameters to tune, such as the number of hidden layers, the size of each hidden layer, what activation function to use, what L2 penalization, etc. etc.

- Sensitive to feature scaling

If some features are not scaled the same as others, they may dominate or be dominated in the output, rendering a portion of the features useless.

- Black box

There is interpretability of the relation between the inputs and outputs. The model just does it's thing and spits out an result.

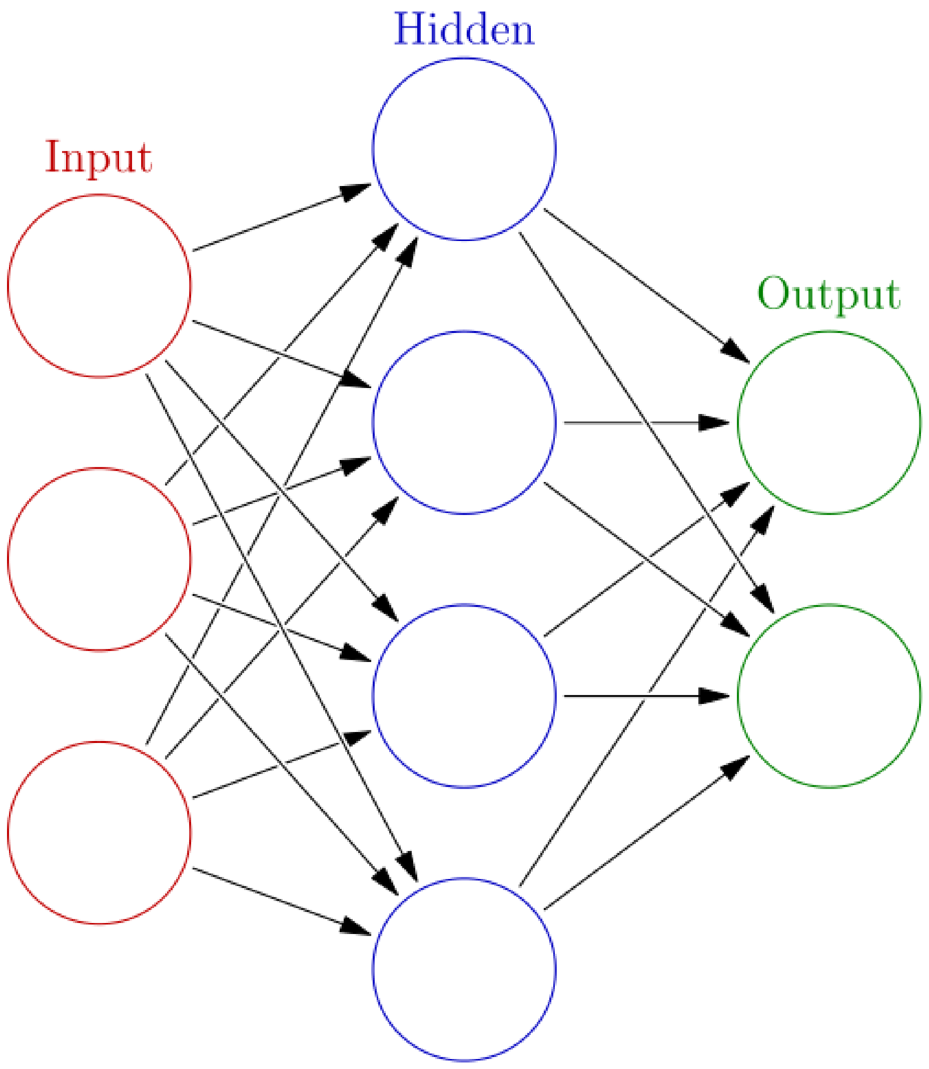

Figure 5.1: Schematic of an Artificial Neural Network source from `https://en.wikipedia.org/wiki/Artificial_neural_network`

Chapter 6

Results

6.1 Time Series Approach

As the methodology to arrive to the final model is the same in R and Python, we will just layout the Python version in detail, and then state the final result using R.

As financial data is not stationary and exhibits non-linear trends, and potentially heteroskedasticity, it is common practice to consider the log returns of the series. That is, to consider the series $X'_{t+1} = 100[ln(X_{t+1}) - ln(X_t)]$. We see that in Fig 6.1 the returns seem to be stationary. We check this by considering the ACF and PACF graphs of the returns.

Looking at Fig 6.2 there are a few lags that fall out of the 95% significance interval. Oddly enough there would seem to be some significant autocorrelation at lag 6, which is surprising as if anything I would have expected there to be lag 7 autocorrelation, as there are 7 observations per day. It's not a very high autocorrelation nonetheless and fitting an ARMA model with seasonality 6 just introduced more structure into the series instead of removing it, so we abandoned that path.

We also checked the normal QQ-plot of the returns (Fig 6.3) and they are clearly not normally distributed. They have heavier tails, so we fitted a Student t distribution, manually tuned to a parameter $\nu = 2.6$ (Fig 6.4). This fits pretty well. At this point one could almost argue that the returns are just white noise with Student t_ν distribution.

However, when we look at the ACF and PACF of the squared returns (Fig 6.5, we notice distinct autocorrelation every lag multiple of 7. This suggest that there is significant structure in the volatility of the returns. This phenomena is called "volatility clustering" and happens a lot in financial Time Series. The idea is that volatility breeds volatility; a period of volatility follows a period of volatility.

To model structure in the volatility of a series is where GARCH model come into play. We first tried fitting a simple ARCH model to the returns, but the

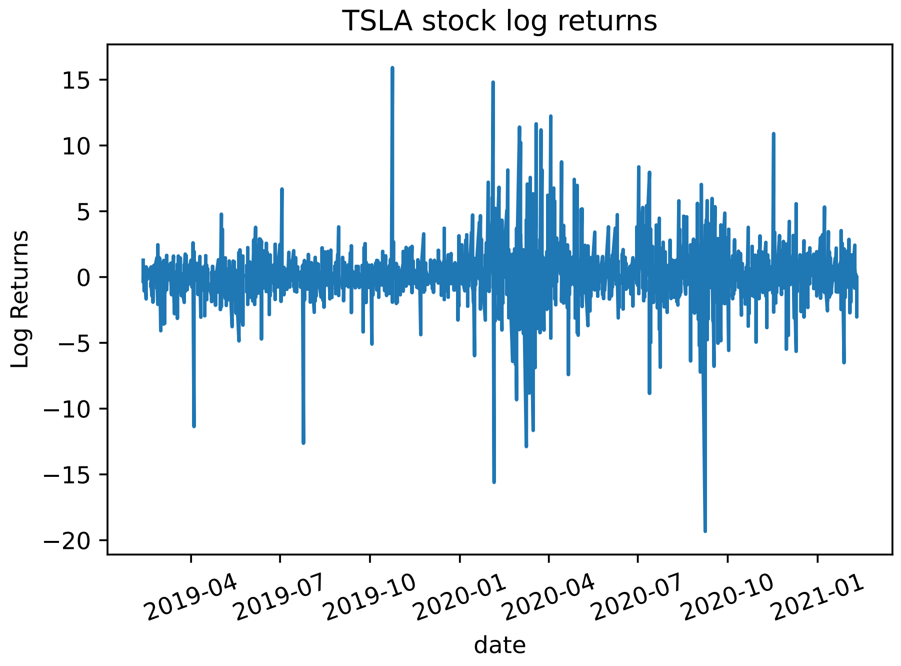

Figure 6.1: Log returns of TSLA hourly opening stock price

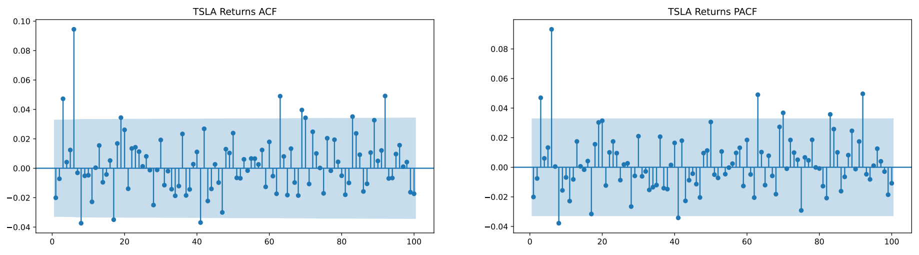

Figure 6.2: ACF plot of log returns of TSLA stock price

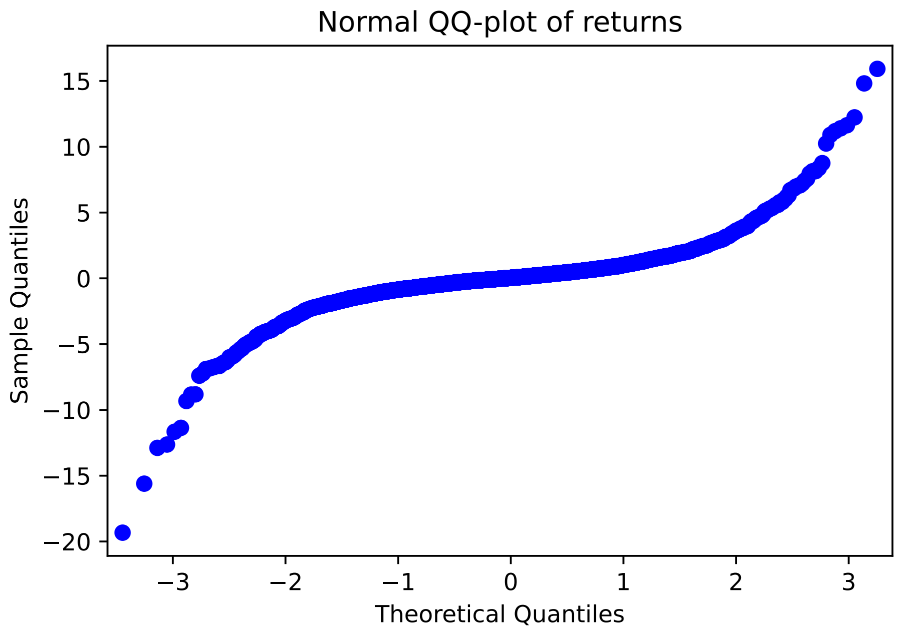

Figure 6.3: Normal QQ plot of log returns of TSLA stock price

results were not particularly better, so then we fit a GARCH(1,1) with Student white noise model to the returns, and as you can see in figure 6.6 and figure 6.7 the fit is very good.

The standardized residuals in figure 6.8 look very much like white noise now. Figure 6.8 also shows the conditional volatility extrapolated by the model. There are several things we see here. First, there is a large increase in volatility between January 2020 and May 2020, which I hypothesize is related to the uncertainty generated by the first wave of Covid-19. The later waves did not affect the volatility much, because by that time the initial panic had subsided, knowledge about the virus and how to fight it was more redly available. But that first wave caused havoc on the markets. The second thing we notice, is that there are spikes of very short duration but high value that happen every so often. They seem to be every 3-ish months and I would hypothesize that they more or less coincide with Tesla's quarterly earnings reports. Indeed, according to Tesla's investors website ... `https://ir.tesla.com/` The dates of the quarterly reports were:

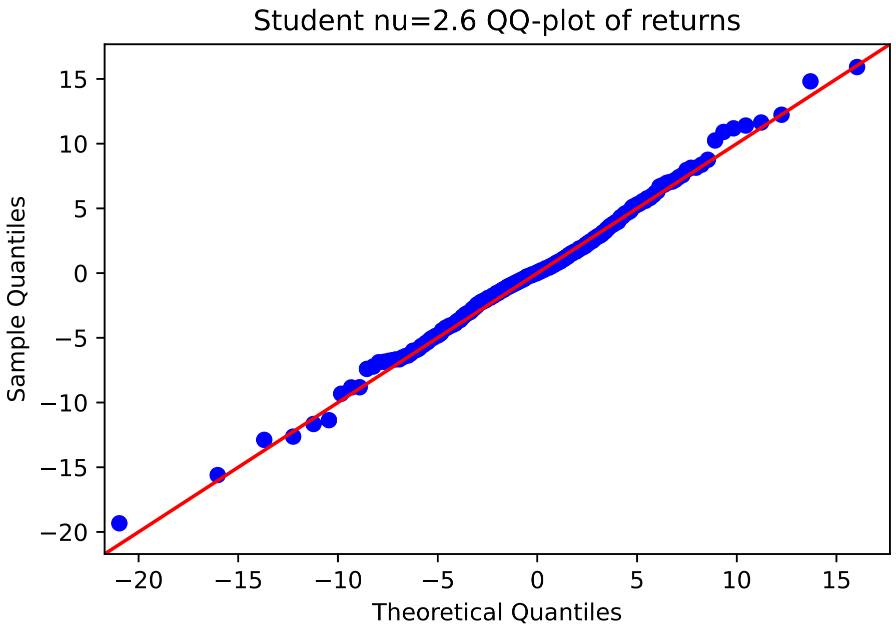

Figure 6.4: Student QQ plot of log returns of TSLA stock price, with a parameter $\nu = 2.6$

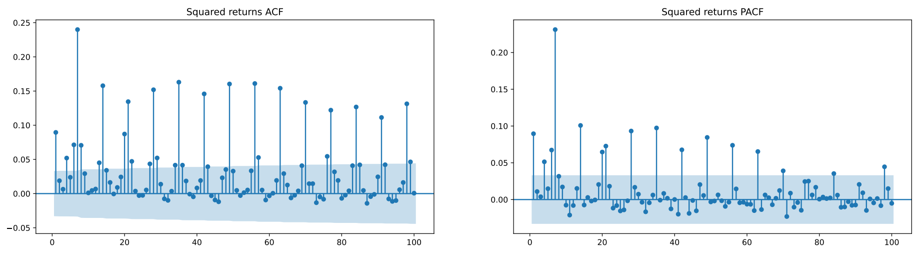

Figure 6.5: ACF and PACF of squared TSLA log returns

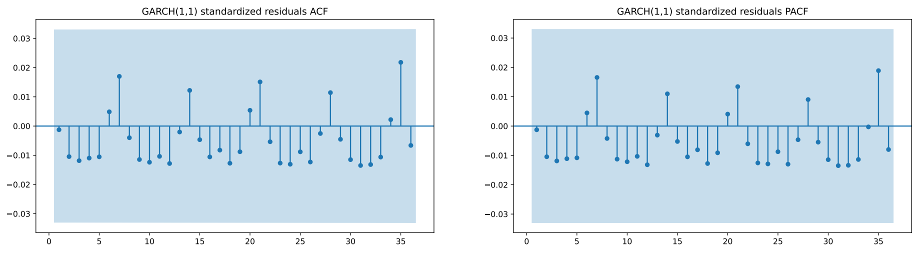

Figure 6.6: ACF and PACF of squared standardized residuals from fitting a GARCH(1,1) model with distribution t_ν

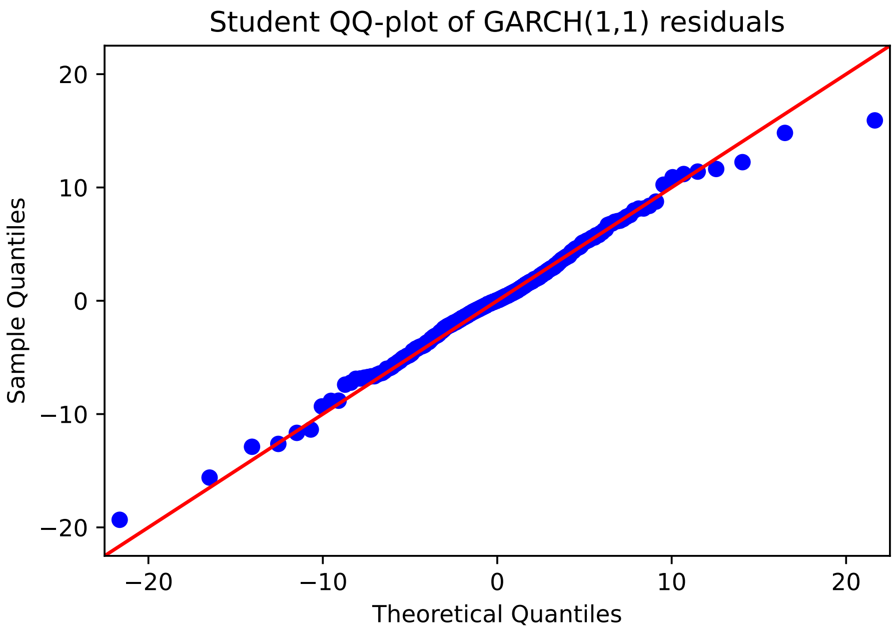

Figure 6.7: Student QQ-plot of GARCH(1,1) fitted to the TSLA returns

Q4/18: Jan 30th 2019
Q1/19: Apr 24th 2019
Q2/19: Jul 24th 2019
Q3/19: Oct 23rd 2019
Q4/19: Jan 29th 2020
Q1/20: Apr 29th 2020
Q2/20: Jul 22nd 2020
Q3/20: Oct 21st 2020
Q4/20: Jan 27th 2021

But this is not quite on point. First there is the case of Q1/20 which doesn't appear to have caused a spike in volatility, but this could just be that is was overshadowed by the Covid-19 first wave. Then there is Q4/20 that doesn't show up at the end of the graph. But as the other spikes are also a bit off from the quarterly report date, it's possible that the spike happens right after the end of the period we are considering, and that the spike are not fully explained by the quarterly report. Perhaps, when the spikes occur before the reports, it is because people are betting they will be good or bad, and there is excitement, maybe because a product launched that quarter or something. And when the spikes occur after the report, it could be because the reports were better or worse than expected, or maybe following a good report, Tesla or Elon Musk announce something big in the following days. What's more, Tesla releases a press release basically 2 weeks prior to the earnings report. For Q4/18, this

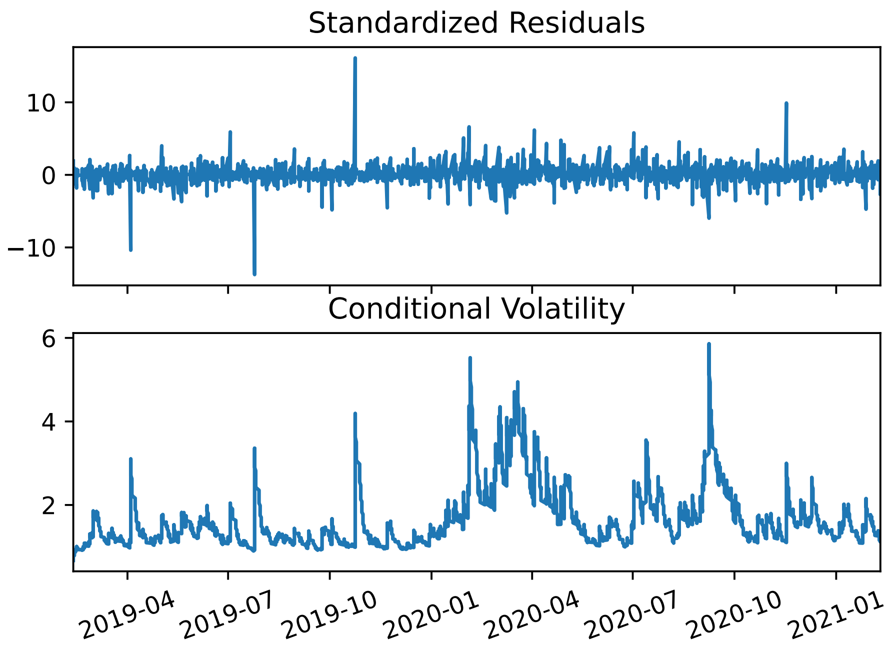

Figure 6.8: Standardized residual of the GARCH fitting process and conditional volatility

corresponds with the spike date.

As for the spike between Q2/20 and Q3/20, this could perhaps be explained by the stock split, which occurred on August 31st and fits relatively well. The first spike in November can be explained by the announcement that Tesla would be added to the S&P500 index. Figure 6.9 shows this.

The final TS model is a GARCH(1,1) model with Student-t distributed residuals. The parameters estimated by Python are:

$$\begin{aligned} \alpha_0 &= 0.042 & confidnceinterval &:[0.003, 0.081] \\ \alpha_1 &= 0.065 & confidnceinterval &:[0.033, 0.098] \\ \beta_1 &= 0.934 & confidnceinterval &:[0.893, 0.976] \\ \nu &= 2.56 & confidnceinterval &:[2.41, 2.72] \end{aligned}$$

The parameters estimated by R are:

$$\begin{aligned} \alpha_0 &= 0.067 \\ \alpha_1 &= 0.137 \\ \beta_1 &= 0.935 \\ \nu &= 2.23 \end{aligned}$$

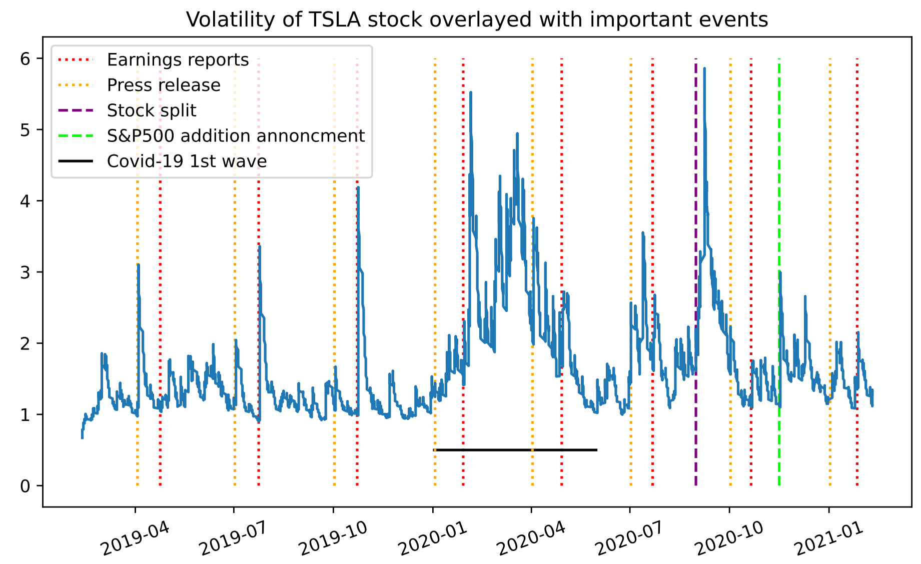

Figure 6.9: Important events overlayed with the GARCH volatility

So there are some slight differences between the R and Python implementations, but the results are close. This could be explained by a difference of initial values for the solver.

Using R, the log returns predictions can be seen in Figure 6.10. The model has mean zero, so it just predicts zero for future values, so that is not very interesting, but what is, is the confidence intervals.

6.2 Machine Learning Approach

As in the TS approach, we will not consider the raw price of the stock, but consider the log returns of the price. As such we are analyzing the variations in the stock price more so than the stock price itself (although indirectly we are). This helps to keep the values contained in a same region of the response space, thus decreasing sparsity problems, while also protecting the first part of the series's contribution, where the absolute variation is small but the relative variation is as large as the later half of the series.

6.2.1 First case

For the first scenario, where we have a univariate response, and use only the external covariates of time, the Google Trends data for the search words "Tesla" and "Musk" as well as Elon Musk's tweets statistics of retweet count and like count, we first standardized the data with a min-max scaler, in order to get all

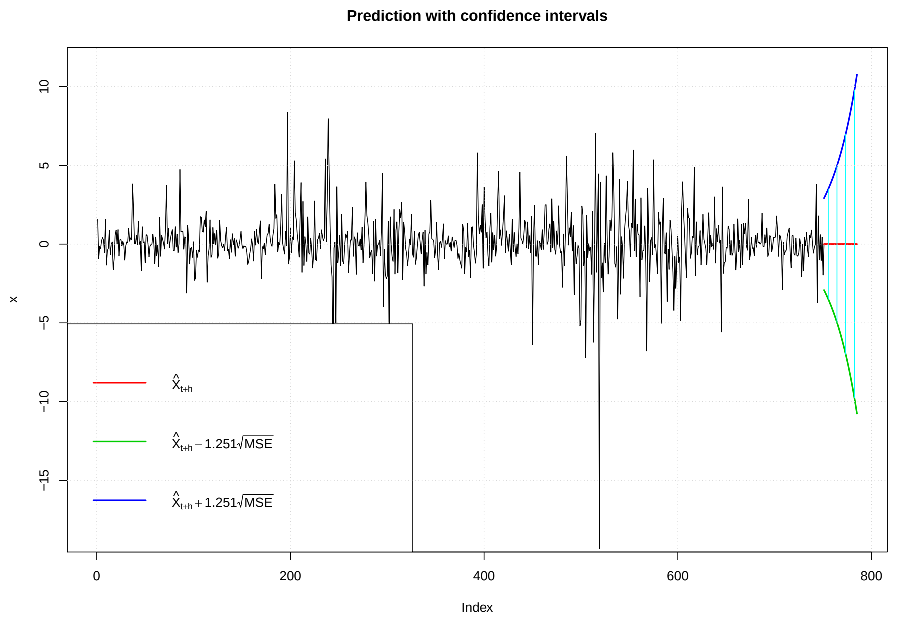

Figure 6.10: Prediction of future returns with 95% confidence intervals

the covariates between 0 and 1. While is is not so important for some of the methods, like Linear Model (though it still is best practice), it is very important for model like ANN, so that one variable with high absolute values compared to others (like for example tweet like counts in the hundred thousands) doesn't overshadow the rest.

After standardizing, we fit 6 different models to the data, with default parameters and a set seed for reproducibility and then used them to predict values in the test set, and compared them with the real values. As can be seen in table 6.1, just looking at the error measures, the Linear Model seems to work the best both if we consider the error on the log return scale (the scale on which the models were fit), and if we transform the results back into the stock price scale. The next best one seems to be the Artificial Neural Network on the returns scale, and is third on the price scale. The second best on the price scale is the Random Forest but it performs second to last on the returns scale. There is a similar phenomena for KNN, which comes in third on the returns scale, but second to last on the price scale. Overall the worst on both scales is the Decision Tree, which is not a big surprise as we saw earlier that a single DT by itself usually performs poorly. Sadly, when looking at R^2, the only model that performs better that just using the expected value, is the Linear Model.

Looking at the plotted prediction versus test values, in figure 6.11, we notice some interesting things. First, only the RF arrives at the right price range at

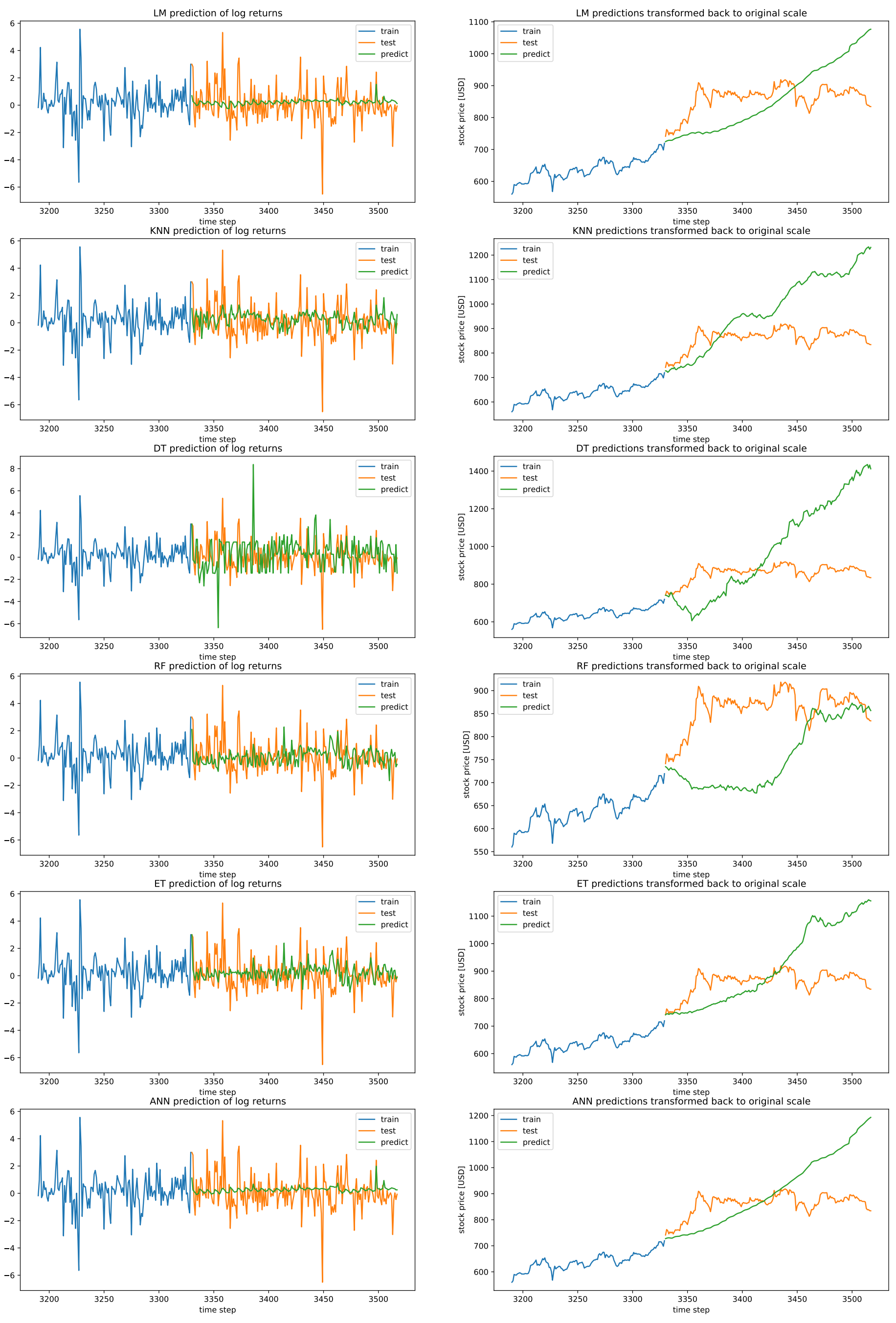

Figure 6.11: Univarite case ML results, on the log returns scale and on the original scale, zoomed into the last part of the series.

Table 6.1: Error measures of the univariate response case, on the log returns scale and original scale

	MSE	R2	MSE_original_scale	R2_original_scale
LM	1.650403	0.008168	9606.311211	-4.8631
KNN	1.808194	-0.086659	31215.174459	-18.051819
DT	3.679189	-1.21106	72790.038001	-43.426554
RF	1.918753	-0.153101	17207.088123	-9.502146
ET	1.878936	-0.129173	20718.301652	-11.645175
ANN	1.671615	-0.00458	18571.14794	-10.334684

Table 6.2: Error measures of the multivariate response case, on the log returns scale and original scale

	MSE	R2	MSE_original_scale	R2_original_scale
LM	2.102195	0.076515	1238.834902	0.557528
KNN	2.970451	-0.304905	11916.125384	-3.256061
DT	3.208266	-0.409377	6911.49391	-1.468566
RF	2.352084	-0.033259	4159.203129	-0.485535
ET	2.525555	-0.109464	3558.157331	-0.270861
ANN	2.352469	-0.033429	2063.433091	0.263007

the end of the test period, even thought the path to it doesn't reflect reality, whereas all the other model predict a continued increase in the price. Second we notice that the output of the LM and ANN are similar, which means the neural network decided that the Linear Model was already roughly the best that could be achieved with the input data. But this could just be because the default parameters of the ANN are not well tuned for this problem. We see also that the model predict the short term values relatively well, say the first day or so, but then perform badly for longer time periods. The DT predicts a sharp change in the price at the beginning, that seems to be roughly of the right amplitude, but just in the wrong direction, so perhaps it detected that there would be something big happening, but didn't know how to decide if it was positive or negative. But there is no way to know this, and it could just be a coincidence.

6.2.2 Second case

For the second scenario, we consider one week ahead of log returns as response variables and consider one month prior of every covariate as feature variable. Just like in the univariate response case, we scale the inputs to fit in the 0-1 interval, using a min-max scaler.

We fit 6 models to the data, using the default parameters and using the same seeds as before for reproducibility. As we can see in table 6.2, again the

LM works the best, followed by the ANN and the two aggregated tree methods, on both scales, and the poorest performing methods are KNN and DT which is not very surprising. Again the LM model performs better than the expected value predictor, according to the R^2 statistic. It is nice to see that this tme the ANN method performed better than the expected value on the price scale, even if less well than the LM method. Again, my hypothesis for this is that LM doesn't require any tuning of parameters, to it is already at optimal output performance, while the other methods require tuning of parameters, and the default parameters are not optimal.

Considering that the aggregated tree methods perform only slightly worse than the expected value predictor, using only the default parameters, with some tuning they could yield nice results. ANN performed decently in this setting, but the performance can vary quite a bit depending on the parameters and also on the seed, so correct tuning would also likely yield good results.

We plotted visually in figure 6.12 the first week of data in the test set, the prediction for the same week, and the month of data used as input to predict it. The fit for the LM looks indeed quite nice, but doesn't manage to capture the final spike. The Random Forest and the Extremely Randomized Trees both capture the general trend, but not it's amplitude. ANN performs a bit better than the trees, but not substantially. All model fail to predict the last spike in the returns (and price).

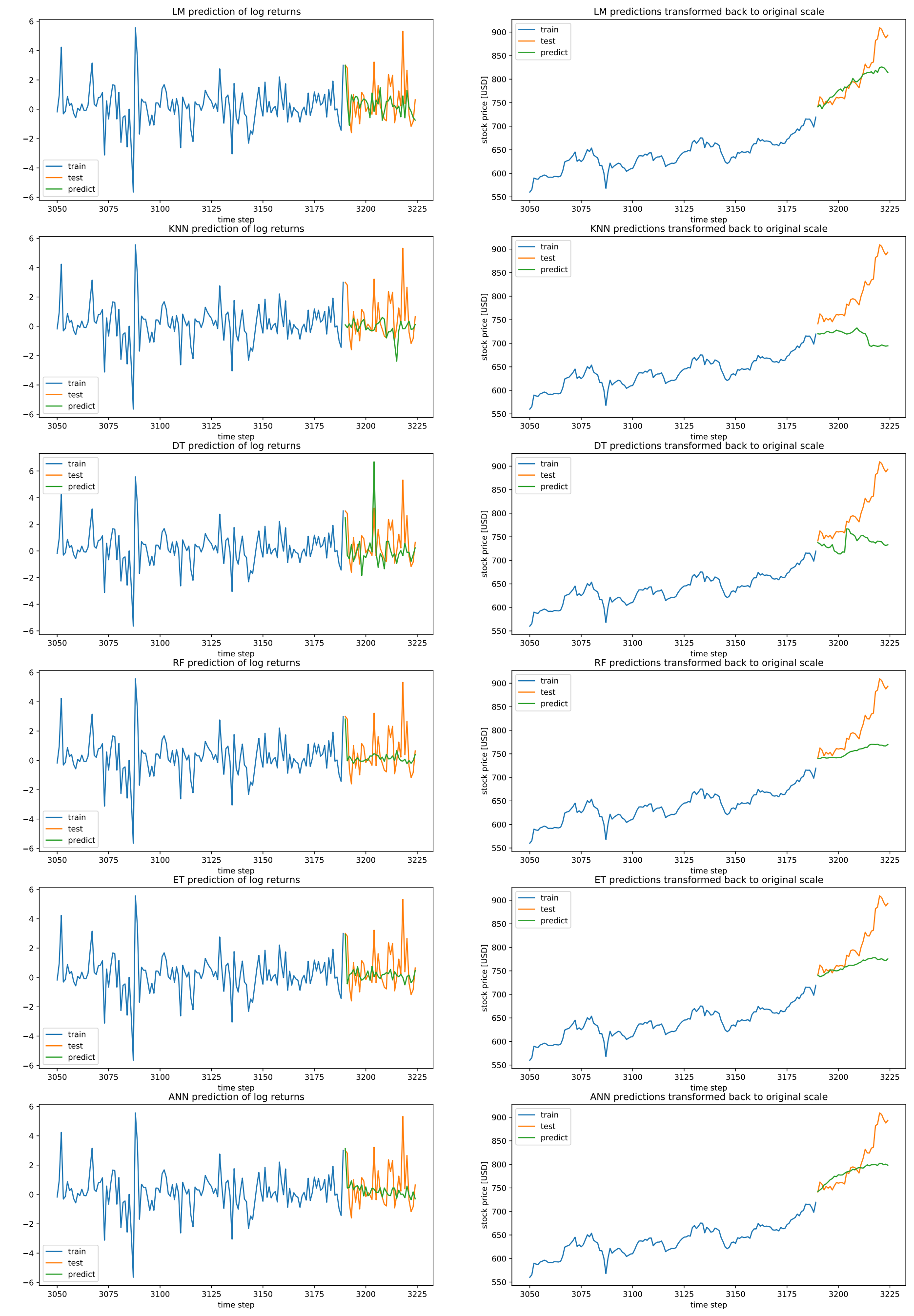

Figure 6.12: Multivariate output case ML results, on the log returns scale and on the original scale, zoomed into the last part of the series.

Chapter 7

Conclusion

The Time Series analysis shows that a GARCH(1,1) model is good at explaining the stock price and it's fluctuations, and give us insight as to why and when the price fluctuates. Other than extreme events, such as the Covid-19 pandemic than created a sharp rise in volatility, there are some events, that occur at given times, know in advance, that create spikes in volatility. For example the quarterly earning reports. These spikes in fluctuation are important, as this is where people make or lose money in large quantities. Being able to consistently predict them, is an asset.

The Machine Learning modeling and prediction shows that for default parameters, simple Linear Model still work the best. However I would like to have tuned the other methods, as this would greatly increase their accuracy and potentially yield better results than the Linear Model. I tried some manual tuning by varying the parameters of methods arbitrarily and received some better results , but didn't spend enough time checking the tuning and being formal about it to include it in this report, out of fear that some results might just be coincidences and might mislead the reader.

We also see from the Machine Learning approach, that the prediction get bad relatively fast after about one day in the future. My theory about this, other than the obvious that the more in the future we try to go, the more uncertainty there is, is the following. The frenzy (measured by Google Search activity and twitter activity) that makes price swing expires a lot faster than one week. My guess is it expires roughly after one day, and in our case, something probably happened 2 days or more after date from which we start predicting, and thus the relevant covariate information was not part of the input. This would explain why non of the models were able to predict the final spike in price.

As far as the comparison of Python and R, we were only able to do this with the Time Series approach, and in this case the results very quite similar, with only minor differences. Explained probably by different choices in initial values or optimizers only, and not due to difference in implementation of methods in general. I was unable in the allocated time to get the R verison of the Machine Learning to work, and so we can not do any comparison for those. I do suspect however that the results would be similar to those of Python as well.

As for a personal stand point, I came into this project with only a little theoretical knowledge about the subject of Machine Learning and one of my goals was to get a practical sense of how it worked and what needed to be done to get them to work. And this was fulfilled. I was surprised how much preprocessing of the data is necessary and how important it is. I also have a better understanding of the methods presented in the report even if not complete yet.

Chapter 8

Future Works

To further this research, I would like to first and foremost tune the models and see what results I get with an optimal or close to optimal tuning. Considering the dates of quarterly earnings report seem to have a big influence on the volatility of the process, as shown by the Generalized Autoregressive Conditional Heteroskedasticity fit, it might be worth taking that into account as a variable. I would also like to find other variables that could be good indicators of volatility more in advance than the Google searches, or tweet reactions. Additionally, I would like to try more novel methods on this problem and see how well they would perform. Long Short-Term Memory models appear to show promise for modeling and predicting financial Time Series, as seen in [7] or [8] And finally , there is the possibility to shorten the prediction interval, as the methods work better on shorter time frames.

Code

All code for this project can be found on `https://github.com/Sekarski/MasterThesis`

Bibliography

[1] G. Box, G. Jenkins, *Time Series Analysis: Forecasting and Control*, San Francisco, Holden-Day, 1970

[2] R. F. Engle, *Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation*, Econometrica, 50 (4), 987-1007, 1982

[3] T. Bollerslev, *Generalized Autoregressive Conditional Heteroskedasticity* Journal of Economics, 31 (3), 307-327, 1986

[4] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, *Classification and regression trees*, Monterey CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1984

[5] L. Breiman, *Random Forests*, Machine Learning, 45(1),5-32, 2001

[6] P. Geurts, D. Ernst., and L.Wehenkel, *Extremely randomized trees*, Machine Learning, 63(1),3-42,2006

[7] Mehtab, Sen. *Stock Price Prediction Using Machine Learning and LSTM-Based Deep Learning Models*, 2020

[8] Ding, Qin. *Study on the Prediction of Stock Price Based on the Associated Network Model of LSTM* International journal of machine learning and cybernetics 11.6 (2020): 1307–1317

Acronyms

ACF Auto-correlation function. 3, 23, 24, 26

AR Autoregressive. 14

ARCH Autoregressive Conditional Heteroskedasticity. 23

ARMA Autoregressive Moving Average. 14, 15, 23

GARCH Generalized Autoregressive Conditional Heteroskedasticity. 3, 6, 15, 23, 25–29

GT Google Trends. 7, 8, 10

KNN k-nearest neighbors. 17, 18

LM Linear Model. 16, 17

MA Moving Average. 14

ML Machine Learning. 4, 6, 15

MSE Mean Square Error. 18, 21

OLS Ordinary Least Squares. 17

PACF Partial auto-correlation function. 3, 23, 26

TS Time Series. 4, 6, 14, 15, 23, 28

TSLA Tesla Stock. 3, 7, 10, 26, 27

Glossary

API tools available for interfacing with a library. 7–9

S&P500 S&P500 index, composed of 500 important US stocks. 28