서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

MINES
Saint-Étienne
Une école de l'IMT

Technical Report

Machine Learning Models applied in Quantitative Finance and Risk Management.

Student: CHRAIBI Salim
Supervisor: KIM Tae-Wan
Internship dates: 30 May 2017 – 28 August 2017

Table of Contents

Abstract:

I did my internship at Seoul National University under the supervision of Mr Kim Tae-Wan, a professor in the Naval department with a strong interest in the new Deep Learning techniques. The most interesting part of this internship is that we had a complete autonomy, we were working with Bouthemy Marin, without any advice just a beginning subject and weekly meetings with the Professor.
Seoul National University is one of the most competitive university in East Asia, enabling it to explore even new subjects and allocate significant budgets to research. My primary mission was to study an article dealing with the concepts of Random Forest, Xg-Boost and their application to financial markets.

Following that, we recoded the article and tried to go further. We have therefore developed an algorithm based on neural networks with R. Then, to improve our knowledge, we began reading articles concerning new techniques on neural networks, until finding the work done by a small London-based company bought by google, DeepMind.

The goal of DeepMind is to "solve the intelligence". To achieve this goal, the company tries to combine "the best techniques of automatic learning and neuroscience systems to build powerful general learning algorithms." The company wishes not only to equip the machines with high-performance artificial intelligence, but also to understand the functioning of the human brain. Thanks to their work, we opened a new field of research, we started with Reinforcement Learning, passing by Q-Learning then Policy Gradient, these techniques made us rethink our model and instead of using a neural network to predict a stock, then thanks to the prediction try to invest, we started coding an artificial intelligence that learn how to trade.

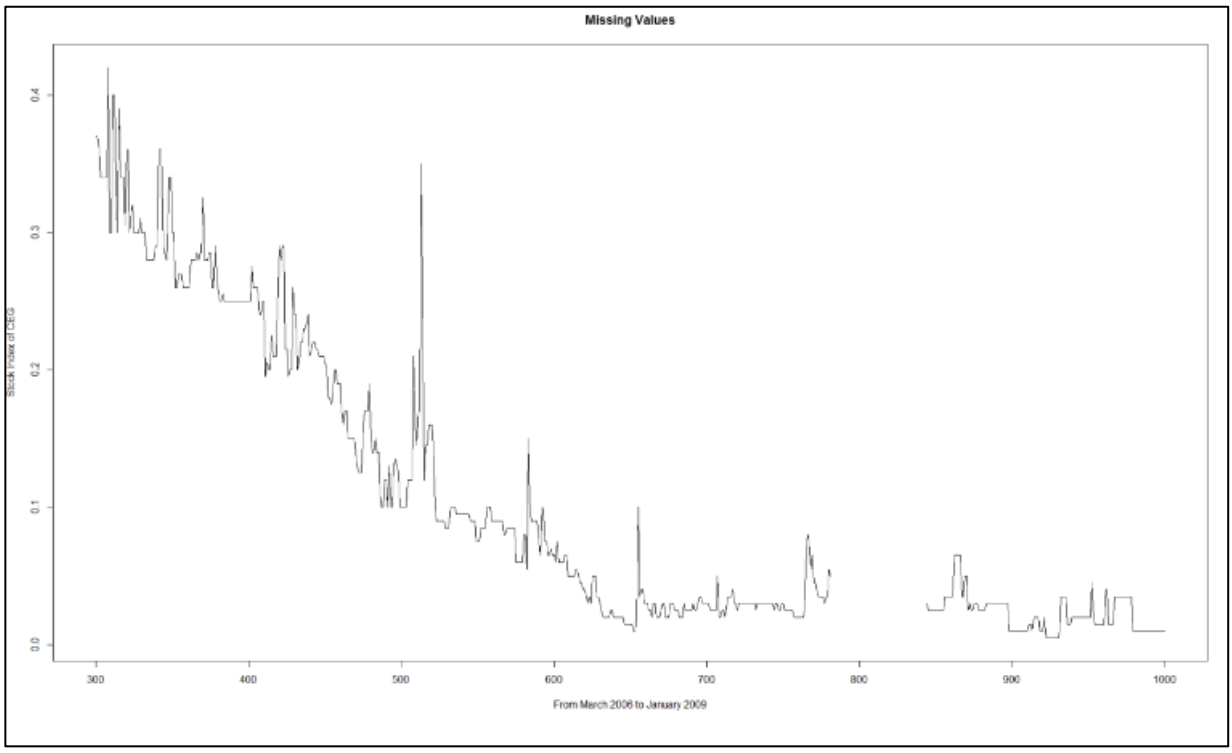
Our artificial intelligence used automatic learning techniques "Deep Learning" with a technique called reinforcement learning, which is inspired by the work of psychologists such as B. F. Skinner in particular on operant conditioning. The technique is called "Deep reinforcement learning". The software learns by performing actions and observing the effects and consequences, in the same way as humans or animals. But until the release of DeepMind, nobody had managed to build a system capable of performing actions as complex as playing a video game. Part of the learning process is to analyse past experiences on several occasions to try to extract more precise information to act more effectively in the future. This mechanism is very similar to those that take place in the human brain.

The second part of our work was made with Python thanks to the Tensorflow backend also deployed by Google. The characteristic of TensorFlow is that it represents the calculations in the form of an execution graph: each node represents an Operation to be performed, and each link represents a Tensor. An Operation can range from a simple addition to a complex matrix of differentiation function. It became, one of the reference frameworks for Deep Learning, used both in research and in enterprise for production applications.
In the following part, we'll explain our work, starting by defining the subject then going through our work on R with simple techniques and then as our understanding of the subject improves we try more difficult techniques.

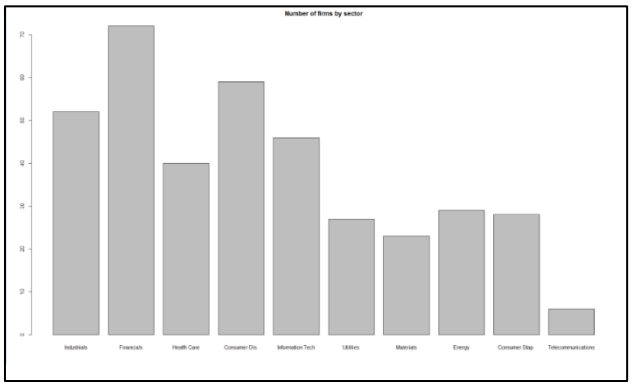
I. Cleaning Data

We have chosen to predict and forecast the S&P 500 Stocks; it contains the best 500 companies in the United States of America in various sectors. First, we have grouped firms according to their predominant sectors (Financials, Industrials, Technologies…), in a total of 11 sectors.
For a period of 15 years, from January 2000 to January 2015; we have predicted then compared to the real values the return in order to determine which models performs well.

Because the composition of the S&P 500 is continuously changing - some firms are leaving or entering the index - our first mission was to clean the data set. Firstly, we have decided to delete some firms with an important number of missing values. If the number of missing values was superior to 10% to the total number of days, then we decided to not take in account this company.



On the other hand, we simply replace the missing values with a linear interpolation between the previous and next known values. Finally, after this cleaning, we obtained a result of 382 stock of firms distributed into 10 specific sectors.



II. Use of different Indicators

In order to get better results and to be able to have an accurate forecasting, we did not only use the close values of the stock but also some indicators which are related to time series.

A- Moving Average (MA)

The first one is the moving average: it reflects the average valuation of a security over a given period.

This indicator gives the average value of the prices over a given period and makes it possible to overcome the aberrations by "smoothing" them. It allows us to have a sense of the trend.
To sense the trend, we have added to the model the moving average for 15 days and 60 days, for the algorithm to understand the long trends and short trends.

B- Relative Strength Index (RSI)

The RSI is a technical indicator invented by **J. Welles Wilder** in 1978. It makes it possible to determine See the power of a trend and indicate whether the market is overbought or over-sold.
An RSI of 50 indicates that the market finds an equilibrium point. When the RSI is greater than 70, the market is said to be overbought and is bidding for a bearish correction. When the RSI is less than 30, the market is said to be over-sold and it is candidate for an upward correction.
Thanks to this we've added the RSI to our model to make the algorithm understand this underlying assessment behind the markets.

C- Average True Range (ATR)

To take the volatility of the market into account we've added the ATR to our model for 14 days and 20 days. The purpose of this indicator is to measure price volatility. **J. Welles Wilder** who was doing a lot of trading on commodities had wanted a tool to filter the high volatility of these markets, traditionally stronger than stocks.
The ATR is not at all interested in the trend followed by the courses but focuses on their volatility.

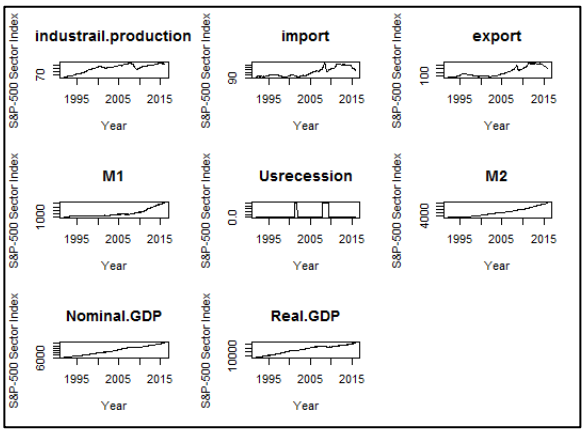
III. Forecasting using Macroeconomics Data

A- Multiple Regression

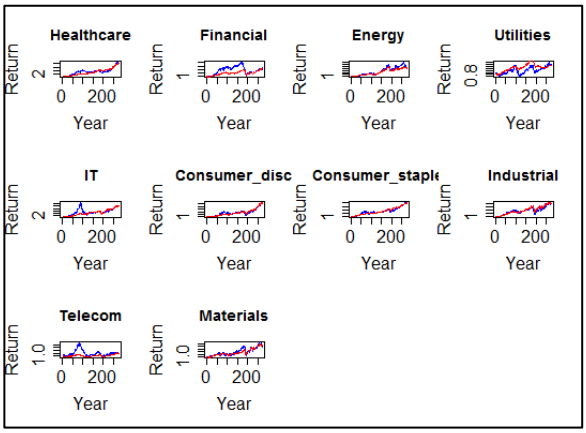
This first forecasting follows the rules of the Efficient Market Hypothesis, because it only relies on underlying information behind the markets. This study will only focus on the sectors. For this part, we're going to introduce the relevant macroeconomics data we'll use:

✓ Federal funds rate: ✓ CPIAUCSL: Consumer Price Index for All Urban Consumers ✓ Unemployment Rate ✓ 5-year treasury rate ✓ Exchange rate ✓ Real disposable personal income ✓ PCE	✓ Industrial production ✓ Import ✓ Export ✓ M1 : Index of Money Supply 1 ✓ Us Recession ✓ M2: Index of Money Supply 2 ✓ Nominal.GDP ✓ Real.GDP

We thought that those data were the most relevant data we can use.



Here are the results of the multiple regressions:



Macroeconomic variables actually do a very good job of tracking and predicting long term trends in the various sector returns - some better than others. But short term price movements are effected mostly by factors other than macroeconomic variables; such as investor sentiment, expectations, earnings releases, information not captured by economic indicators and, of course, randomness.

IV. Forecasting using ARIMA Models

A- Definition

In statistics and econometrics, and in particular in time series analysis, an autoregressive integrated moving average (ARIMA) model is a generalization of an autoregressive moving average (ARMA) model. Both of these models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting).

The ARIMA models are bases on an aggregation of Auto-Regressive Model (AR), and Moving Average Models.

B- Method

Testing Stationarity:

Firstly, we have to check if the series can follow an ARIMA model by testing the stationarity. We will test the stationarity of our series using the **Augmented Dickey-Fuller unit root test**. To make our series stationaries, we'll convert them thanks to a differencing method; the differenced values from a new time series can be tested to reveal new correlations or interesting statistical properties.

Finding the parameters:

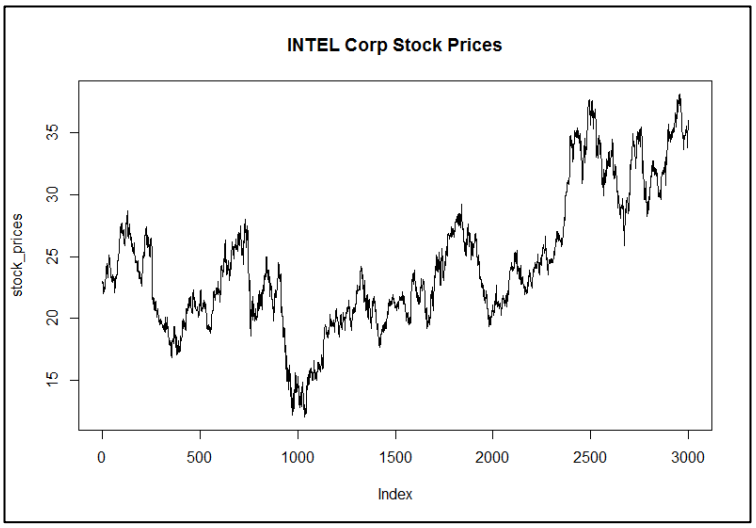
The ARIMA models are based on two types of models, the Auto-Regressive (AR) and Moving Average (MA).To find the parameters, and we will use Autocorrelation function (ACF) and Partial Autocorrelation function (PACF). Thanks to these two functions we are able to find the **right MA and AR parameters**.

Forecasting:

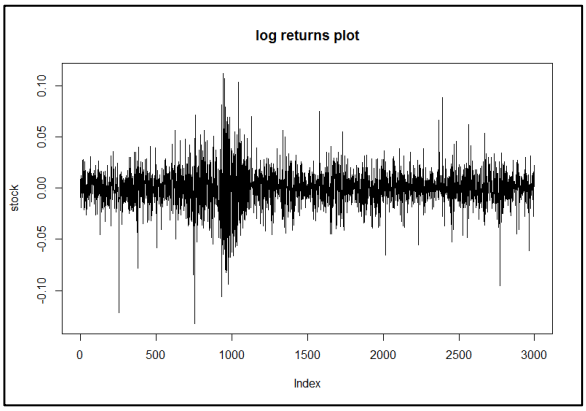
After having determined the right parameters, we will test the accuracy of our model thanks to training data and then we can compare the prevision with the real data set.

C- Applying

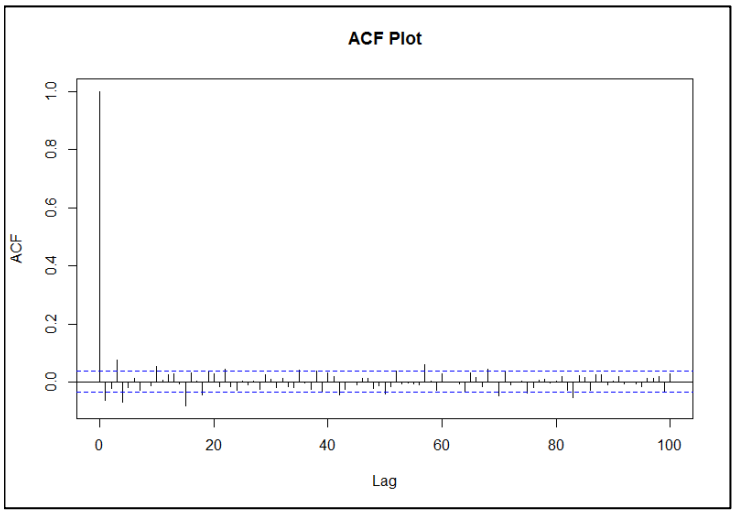
As we have already extracted all the data related to SP500 stocks, we will focus only on the INTC stock, corresponding to Intel Corp, because it's a relevant stock that correlate with the whole SP500.

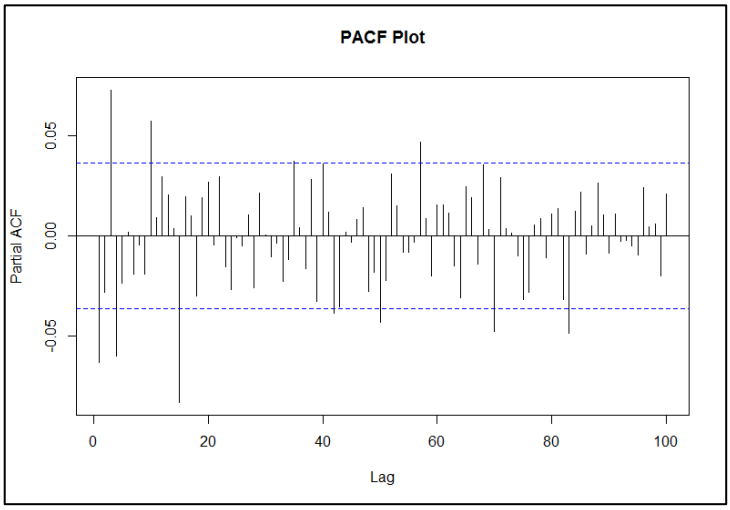


In the next step, we compute the logarithmic returns of the stock as we want the ARIMA model to forecast the log returns and not the stock price. We also plot the log return series using the plot function.

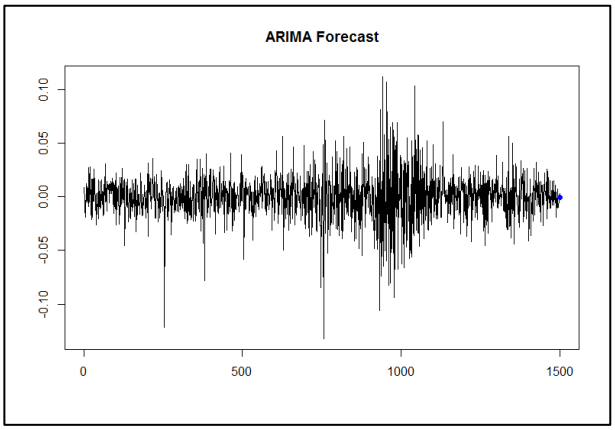


Then, we test the stationarity, the Dickey and fuller test tell us that p =0.01, so the logarithmic returns of the stocks are stationaries. Thus, we can perform the ACF and PACF.

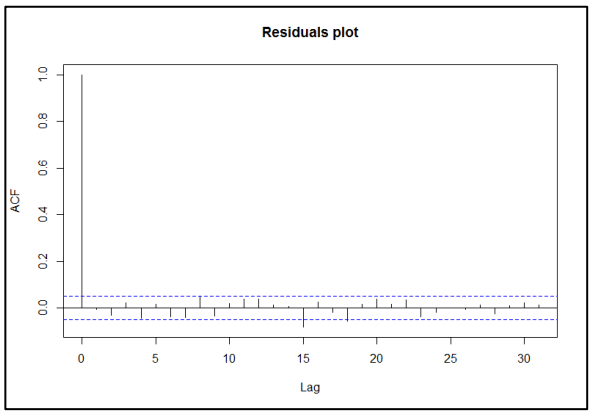




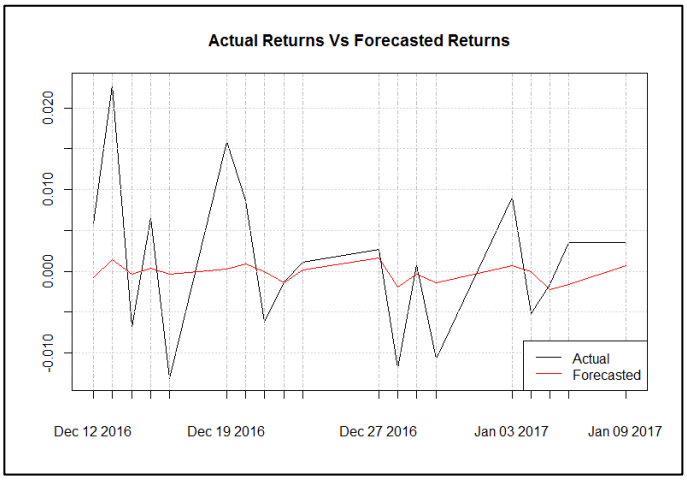
From these plots let us select AR order = 5 and MA order = 3.
Thus, our ARIMA parameters will be (5, 0, 3). Thanks to the ARIMA model, we forecast the last return point.



Then we plot the residuals and see that the model is accurate:



Then we compare the actual returns and the forecasted returns:



V. Trend Prediction using Extreme Gradient Boosting

XGBoost is the short word for "Extreme Gradient Boosting", where the term "Gradient Boosting" is proposed in the paper *Greedy Function Approximation: A Gradient Boosting Machine*, by Friedman.

A- Method

Extreme Gradient Boosting (XGBoost) is similar to gradient boosting framework but more efficient. It has both linear model solver and tree learning algorithms. So, what makes it fast is its capacity to do parallel computation on a single machine. This makes XGBoost at **least 10 times faster than existing gradient boosting** implementations.

Since it is very high in predictive power but relatively slow with implementation, "XGBoost" becomes an ideal fit for many competitions. It also has additional features for doing cross validation and finding important variables. There are many parameters which need to be controlled to optimize the model.

The objective of the XGBoost model is given as:

$$\mathbf{Obj = L + \Omega}$$

L is the loss function which controls the predictive power, and
Ω is regularization component which controls simplicity and overfitting

As XGBoost, only use numerical arguments, we'll convert our data to numerical matrices, and make our predictions on these matrices.
We split our data on two different sets and we try to predict data according to these subsets.

B- Applying

We apply the XGBoost algorithm to our train dataset. We will create 243 epochs for the error to converge:

```
# Train the xgboost model using the "xgboost" function
dtrain = xgb.DMatrix(data = X_train  , label = Y_train)
xgModel = xgboost(data = X_train,label = Y_train, nround = 243, objective = "binary:logistic")
```

```
[231]   train-error:0.151794
[232]   train-error:0.151794
[233]   train-error:0.151794
[234]   train-error:0.151794
[235]   train-error:0.151794
[236]   train-error:0.151794
[237]   train-error:0.151794
[238]   train-error:0.151309
[239]   train-error:0.151309
[240]   train-error:0.151309
[241]   train-error:0.151794
[242]   train-error:0.151794
[243]   train-error:0.151309
```

Then we use cross-validation, to estimate the error made.
In this case, the original sample is randomly partitioned into n-fold equal size subsamples. Of the n-fold subsamples, a single subsample is retained as the validation data for testing the model, and the remaining subsamples are used as training data.

The cross-validation process is then repeated n-rounds times, with each of the n-fold subsamples used exactly once as the validation data:

```
# Using cross validation
cv = xgb.cv(data = dtrain, nround = 243, nfold = 20, objective = "binary:logistic")
# Make the predictions on the test data
preds = predict(xgModel, X_test)
```

```
Console ~/
[231]   train-error:0.153760+0.002228   test-error:0.500939+0.042004
[232]   train-error:0.153504+0.001664   test-error:0.501909+0.042860
[233]   train-error:0.153428+0.001644   test-error:0.502395+0.042393
[234]   train-error:0.153351+0.001651   test-error:0.502862+0.042989
[235]   train-error:0.153326+0.001635   test-error:0.502862+0.042769
[236]   train-error:0.153377+0.001660   test-error:0.504318+0.043087
[237]   train-error:0.153326+0.001594   test-error:0.505289+0.043415
[238]   train-error:0.153198+0.001516   test-error:0.505774+0.042474
[239]   train-error:0.153224+0.001528   test-error:0.505289+0.043088
[240]   train-error:0.153121+0.001575   test-error:0.504327+0.041999
[241]   train-error:0.153045+0.001543   test-error:0.503842+0.041482
[242]   train-error:0.152994+0.001539   test-error:0.503842+0.042046
[243]   train-error:0.152866+0.001601   test-error:0.502395+0.043164
>
```

After that we compared with our data, and plotted the binary prediction. In this stock example, the binary prediction is:

[0 0 1 1 0 1]

For the first and the second day predicted, the price will decrease, and then it will increase for the next two days.

After, discussing this output, we have printed the error made by our model, and we found that the average error made between our prediction and the test data is near 0.5: this model was not really relevant.

To have better performance, various economic data can be added to the main data.

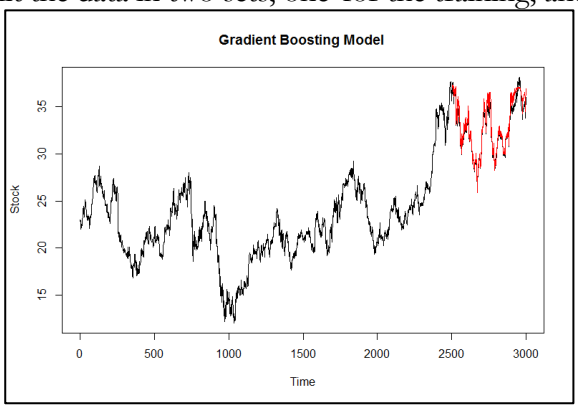
VI. Forecasting Stock Returns with Extreme Gradient Boosting

A- Prediction

Here we use the same stocks, but instead of forecasting only the trend (up or down), we will use the whole S&P500 data to predict the Intel Corporation Stock Price.

First, we convert our raw data, to a matrix, and then we split the data in two sets, one for the training, and the second one for the testing part.

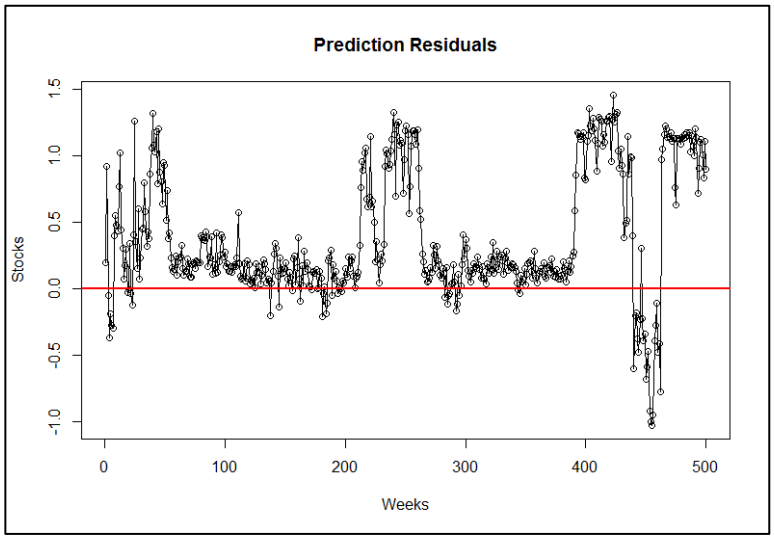
And here is the result of the day-by-day forecast:



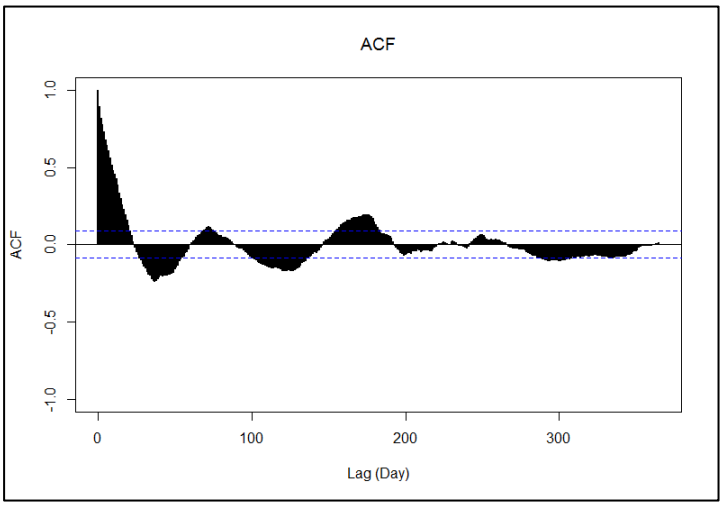
To make this prediction, we have used more than 1000 rounds, but the root mean square error tends to converge for 820 rounds
The accuracy of our model is near 95%.

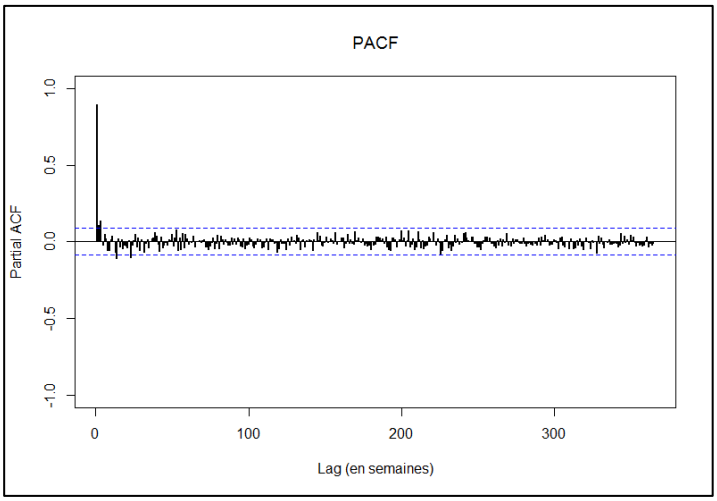
B- Residual Analysis

As the accuracy of our model tends to be high, we will study the distribution of the residuals in order to see if the model seems relevant:



We see, that the residuals tend to be near 0, we will try to find if it is a Gaussian white noise. To find it out, we are doing an ACF and PACF.





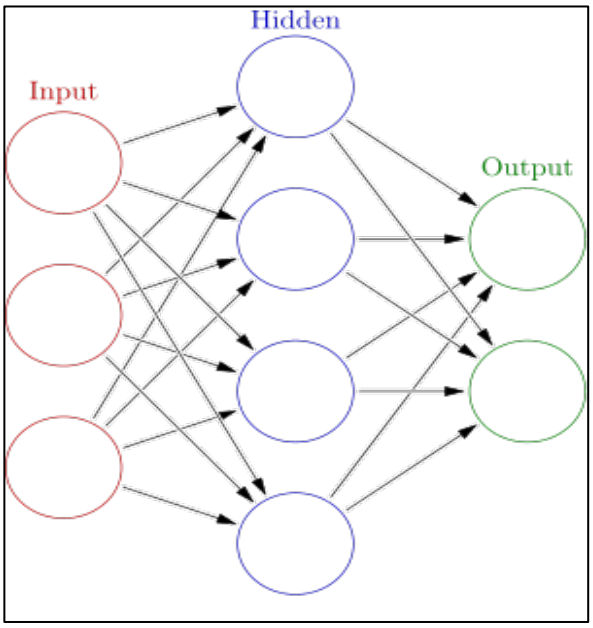
The lagged ACF and PACF for 365 days are between -1 and 1, we can then conclude that it is a white noise: our predictions are quite relevant.

VII. Deep Learning: Recurrent Neural Networks Forecasting

Neural networks are not recent in fact; they were first introduced in the 1950s.

However, they fell into disuse in the 1970s because they were **too expensive to calculate**. They became a subject of interest in the 90s and are now experiencing their glory thanks to the deep architectures (Deep learning) and to the **computing power** proposed by GPUs and CPUs nowadays.

The goal of a neural network is to **mimic the human brain**: in inputs there are some data (in our case this is the close values of the last two weeks) and the neural network will give us in outputs the prediction of the close value for the following day.



The hidden layer is composed of **many cells with a weight associated** to each. Firstly, the weights of the network are initialized randomly and then they are adjusted during the training phase.

At the **end of the training phase**, we obtain a neural network which can predict with a good approximation the output value according to the close values given as inputs.

We are going to use a special category of neural network: the **recurrent neural network**.

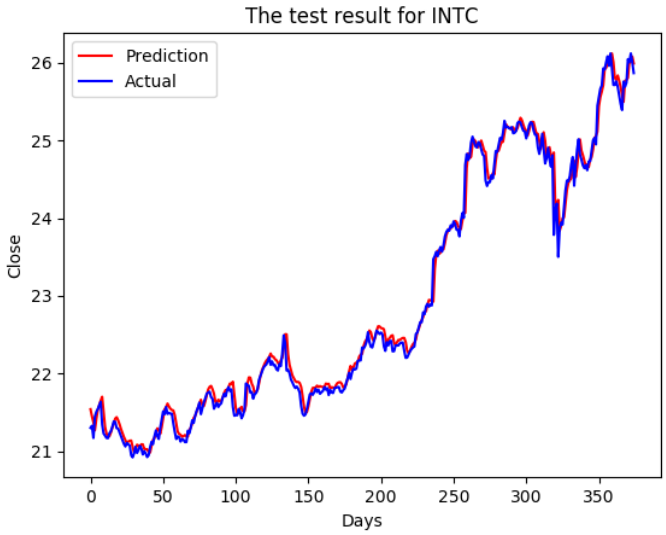
Recurrent neural networks (RNN) is a type of neural network which works with sequences: it allows to process sequential data, a sequence of inputs $[\mathbf{x_0}. . . \mathbf{x_m}]$. At a time **t** they calculate their output as a function of the input $\mathbf{x_t}$ but also of the **state of the layer hidden at the previous time**.
In this way, it is possible **to keep the temporal dependency of the sequel**, in our case it can be applied to a following sequence of close values.

The basic idea is to keep in memory the sequence of inputs [$\mathbf{x_0}$. . . $\mathbf{x_m}$] and its order because we are using the last following close values to predict the new one.

In fact, recurrent neural network are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn, and that's something **useful to work on time series**. As a rule, algorithms exposed to more data produce more accurate results.

A- Applying

In the following example, we are using the **stock value of the Intel Firm**. Firstly, we'll normalize the Closing Price, and then run a recurrent neural network on it. Each day, we are going to predict and plot the **following day based on the last two weeks**, we are doing it for the last year.



We can see that the model tends to fit the real value, the Mean Squared Errors tends to be really low, which means that the prediction is really close to the actual value for a day-by-day prediction.

Train Score: 0.00017 MSE (0.01 RMSE) Test Score: 0.00004 MSE (0.01 RMSE)

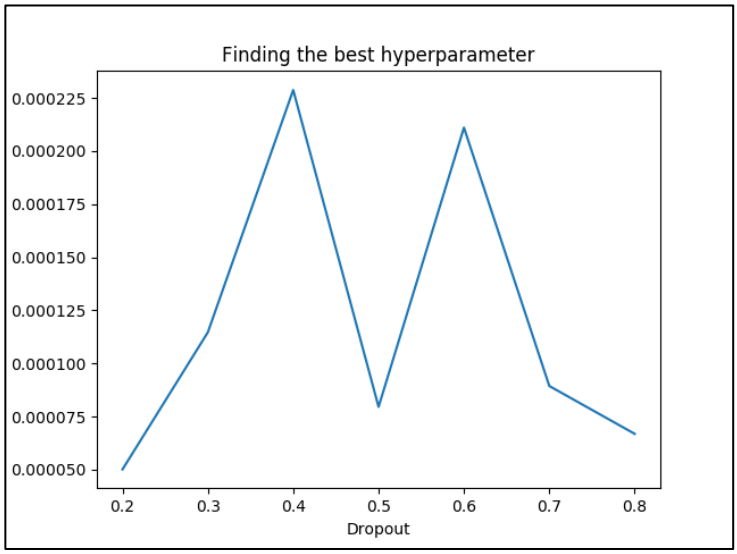
As preconized by Christopher Krauss (2016), "careful hyper parameter optimization may still yield advantageous results for the tuning-intensive deep neural networks". This is the reason why we are looking for the best parameters of the neural network to obtain the best forecasting model.

B- Tuning Hyper Parameters

1- Optimal Dropout value

Firstly, we are going to carry about the dropout value. The dropout parameter indicates the number of cells that the neural network is going to forget every pass in order to avoid overfitting. If the number is 1 then all the cells are forgotten each pass and if it's 0 we keep all the cells.

We'll try different Dropout values = [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8], and find the one with the lowest difference between the prediction and the real value.

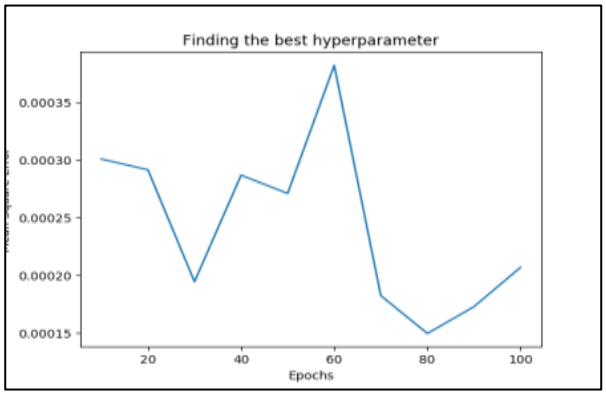


We've run our model with different dropout values, and then evaluated the Mean Squared Error of each dropout value, and concluded that d = 0.2 was the best one.

2- Optimal number of epochs

The epoch is the pass to train the neural network. Actually to correctly adjust the weight of the neural network, we have to run the algorithm over the same train model a lot of time in order to **the weight of the network tend to adjust** and slightly convey to their final value.

We'll try different epoch values between 10 and 100, and evaluate the Mean Squared Value of each epoch value.

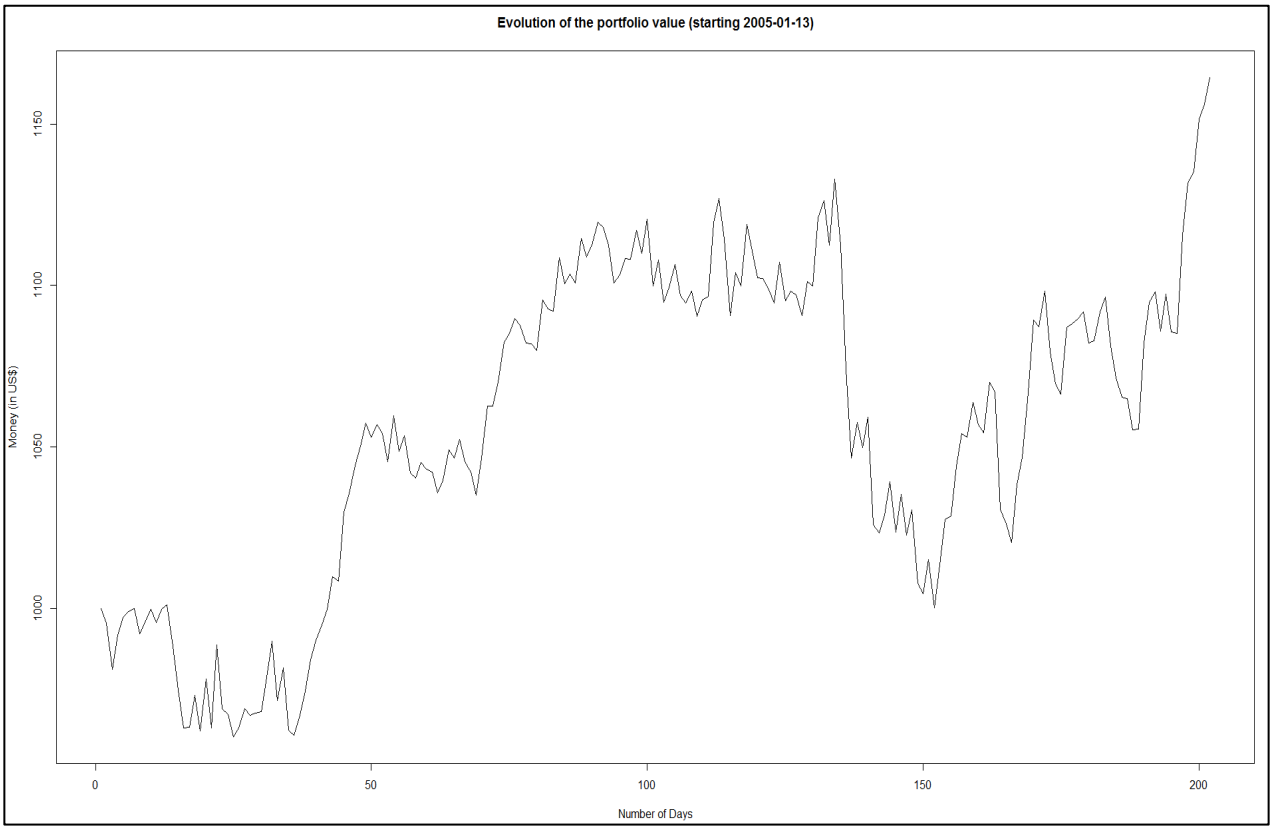


We found that the best number of epochs is 80.

C- Simulation of a portfolio

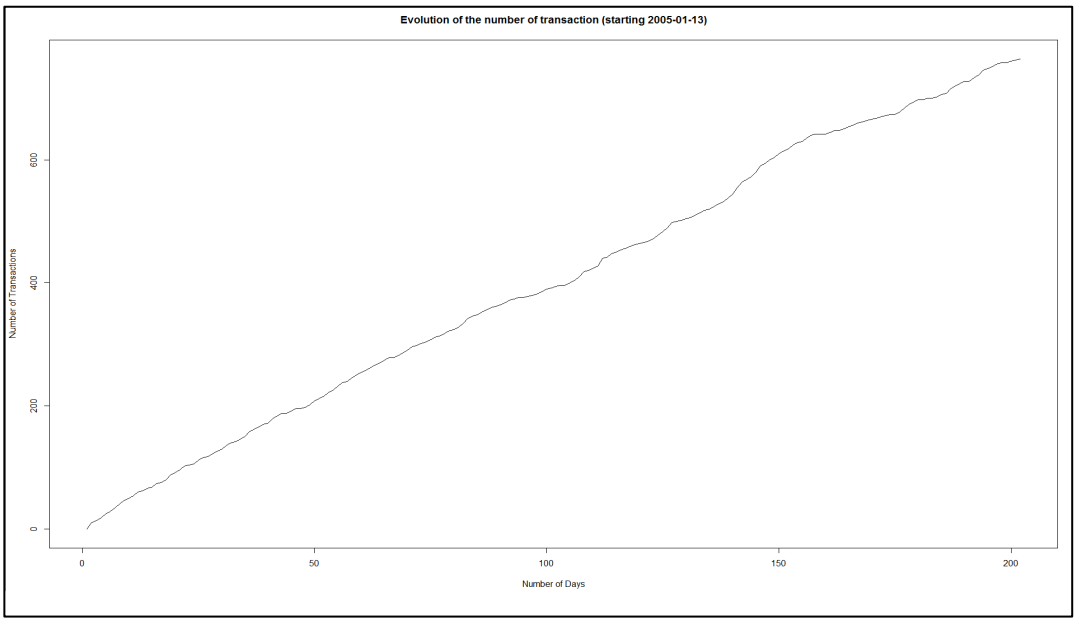
In order to confirm our forecast, prediction and to calculate the return over a large period, we have decided to **simulate a portfolio of the S&P Firms**.

After cleaning and deleting some missing data, we have obtained a portfolio which contains an average of **350 companies from the S&P 500 Index over 10 years**. Our trading strategy is thus the following: everyday, we predict the return of each firm and we sort them by values. Then, we decide to buy the top 10 firms and to sell the worst 10 firms among the firms inside our portfolio: **each day we hold exactly 10 actions of firms, and the quantity of firms buy and sold is the same**.



By operating this strategy, we at the first, aim to perform a day-by-day strategy and gain small amount of money each day. Unfortunately, we found this strategy did not perform very well because of two reasons.

Firstly, because of the transaction cost, a daily trading tends **to increase the number of transactions and thus reduce the benefit.**



Secondly, this strategy aims to gain small amount of money each day, but because we are here looking at the evolution of **stock over a period of 10 years**, it might be more rewarding to us to reduce the number of transactions and to space out our predictions. Moreover, some firms may have doubled or tripled their value in years, and then it is more **interesting to perform a long term strategy**.

VIII. Deep Reinforcement Learning

After reading documentations about trading strategies, we thought that implementing various trading strategies was not the perfect way to enhance our models so we started to **explore other methods and found that Reinforcement Learning applied to trading strategies was really interesting**.

The concept behind this is that we consider an agent, we just give him the raw financial data and he has to **learn how to trade by himself** and find the best trading strategy this way.

This kind of technique is called reinforcement learning: the computer is learning thanks to rewards. We can assume that our agent is just like a little child that has **reward** when he does something **good**, and have a **punition** when he does something **bad**, and as he grows up, he learns how to behave.

Learning by reinforcement refers to a class of automatic learning problems; the aim of these is to learn from **experiments what to do in different situations**, to optimize a quantitative reward function.
A classical paradigm for presenting reinforcement learning problems is to consider an autonomous agent, immersed in an environment, and making decisions based on its current state. In return, the environment provides the agent with a **reward, which may be positive or negative.**

The agent seeks, through iterated experiments, an **optimal decision-making behavior** (called strategy or policy, which is a function associating the current state with the action to be executed) in the sense that it maximizes the sum of Rewards over time.

Formally, the basis of the reinforcement learning model consists of:

- ✓ A set of S states of the agent in the environment;
- ✓ A set of actions A that the agent can perform;
- ✓ A set of scalar "reward" values R that the agent can obtain.

At each time step t of the algorithm, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$.
It chooses an action $a \in A(s_t)$ and receives a new state from the environment s_{t+1} and a reward r_{t+1} (which is null most of the time and is classically 1 in some key states of the environment).

Based on these interactions, the reinforcement learning algorithm should allow the agent to **develop a policy** $\Pi: S \rightarrow A$ which allows it to maximize the amount of rewards.

The latter is written $R = \sum_0^n r_i$ Markov decision processes (MDP) that have a terminal state, and $R = \sum_0^n r_i \gamma^i$ for non-ending MDPs (where γ is a devaluation factor ranging from 0 to 1 and allowing, according to its value, to take into account the rewards more or less far in the future for the choice of actions of the agent).

Thus, the reinforcement learning method is particularly suited to problems that require a compromise between the quest for short-term rewards and long-term rewards.
We'll try to apply this technique to a portfolio. Instead of testing different trading strategies, we'll let the **algorithm learn how to trade and find the best solution.**

A- Q-Learning

The Q-learning is a reinforcement learning class requiring no initial model of the environment. This learning method can be applied to find a sequence of actions associated with states of any (finite) Markov decision process.

This works by learning a state value function that allows us to determine the potential benefit (reward) of taking some action in a certain state by **following an optimal policy.** The policy is the rule for selecting the successive actions of an agent in the current state of the system.

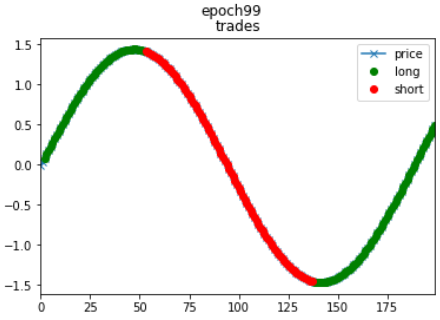
When this action-state value function is known, the optimal policy can be constructed by selecting the **maximum-value action for each state**. This can be resumed with the Bellman's equation:

$$Q(s,a) = r + \gamma(max(Q(s',a'))$$

With r the reward of the action performed, gamma the discount factor (i.e. if it's close to 1 then we hope for the best reward in the future) and Q is the function that given a state and an action will give a value associated.

With this equation is it possible to find the best actions to perform at each time in order to obtain the best rewards in the future. Unfortunately, we don't know this function so we **will use a neural network to approximate it**; this is the essence of the Deep Q Learning.

In the beginning we started with a cosine function (very simple). The goal was not to find the best trading strategy but just sell when the price was decreasing and buy when the trend was going up.
We obtained:

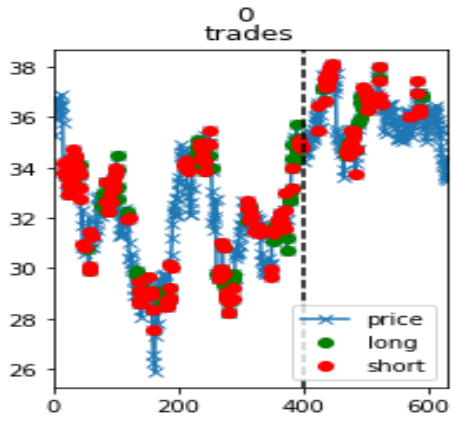


After implementing that, we tried to use the same procedure with stock value. We've used raw data from 01/01/2015 to 07/07/2017.

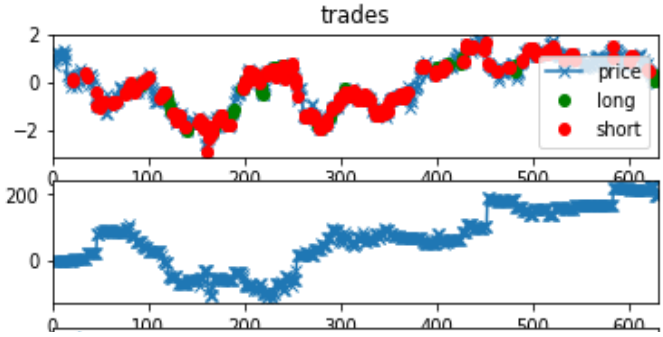
In that case, the **reward was only the money obtained** (or lost) after performing each action.

For example, if the trend was going up and the algorithm was buying then it won some money. It was the best **action to perform and it got a reward** for that, next time it will probably do it again.

On the other hand, if the trend was going up and the action was selling then the reward (portfolio value) would be negative: the computer **will learn to forget this action and next time it will perform another one**.



In this first graph, we see how the algorithm is learning, then we plot the 99 other plots and see that step by step, the algorithm learns how to trade by itself.



B- Double Q-Learning

In some cases, the Q-learning is not very efficient because it **tends to over-estimate the q values** given by the algorithm. In fact, the max operator in the standard Q learning uses the same values to evaluate and also **to select which action** it should perform.

The basic idea of **double Q learning is to use two neural networks and two set of weights.** The first neural network will determine the greedy policy (i.e. which action should we apply) and the second neural network will give us the **value associated**.

In fact, the **recurrent formula** of the Q-value can be written as this one:

$$Y_t^{\mathrm{Q}} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t)\,.$$

To implement the double Q-learning, we inspired from the work paper of *Hado van Hasselt* (2010) where the algorithm is the following:

Algorithm 1 Double Q-learning

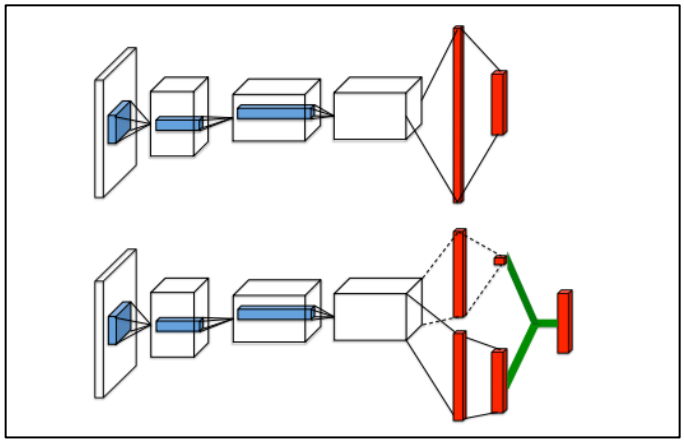
Initialize Q^A,Q^B,s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

With this model added, we have observed that the Q values are less important and tend to be smooth. Furthermore, the speed of the algorithm **has slightly increased** and we have obtained better results with many stocks.

C- Dueling Network applied to Double Q-Learning

Another possible improvement of the Q-learning algorithm is the utilisation of a duelling network instead of a simple network. The idea is based on the **decomposition of the Q function,** Q(s,a) is the function that give us the information of how it is good to take an action given a certain state.
This function can be decomposed as following: Q(s,a) = A(a,s) + V(s)

Where the V function is the value function, it tells **us how good the current state is**. On the other hand, the A function is the action function, and it informs of the **value of the action** to be in another state.



The improvement is the following, one part of the neural network will be used **to approximate the V function** and will give us the V value and the other part will approximate the action function and will be used to **output the actions vector**.

After that, we just have to combine the two in order to obtain the Q function, but it is not possible to just do the aggregation:

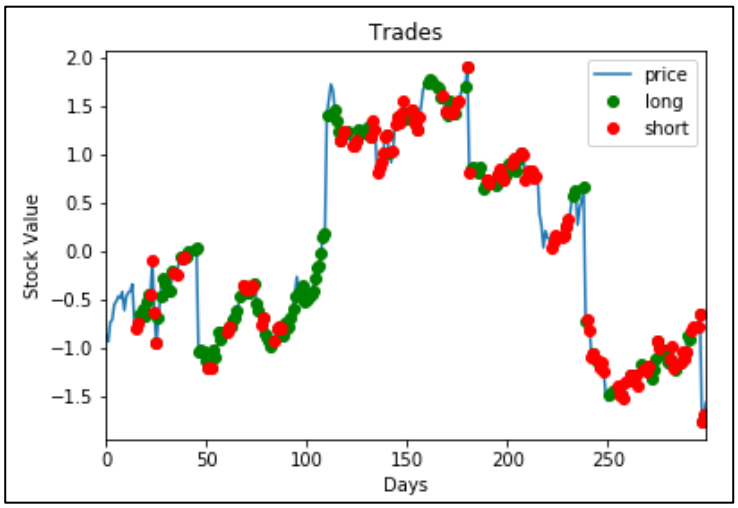
$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha),$$

With theta, alpha and beta the weight of the layers used to approximate the functions. In this case, we shall keep in mind that it is only an **approximation and the values given by the networks might be over-estimated**.
In order to avoid this phenomenon, the aggregation formula is slightly different and we use the mean to **adjust the value of the Q-function.**

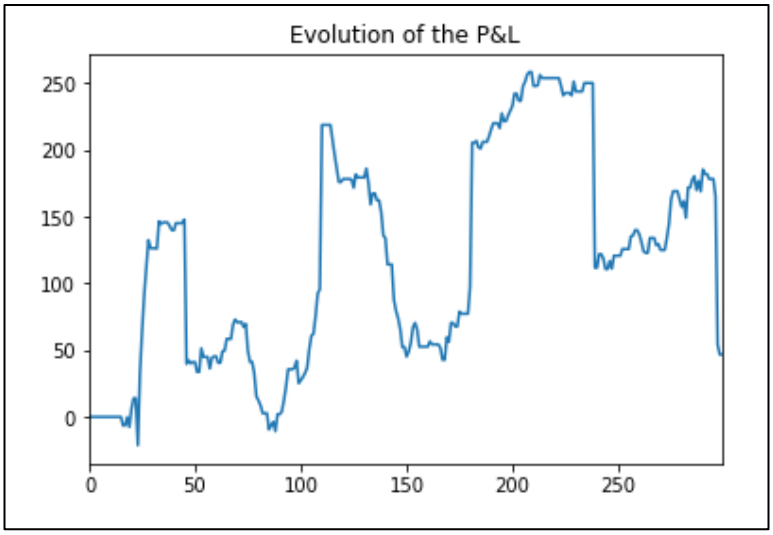
$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right).$$

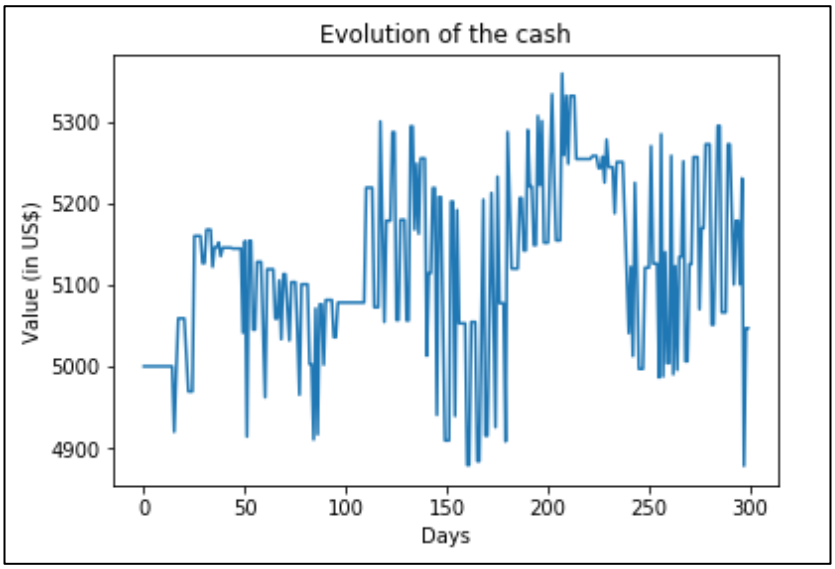
IX. Results and Conclusion

For example, down below we can find the result for the stock value AKAM (Akamai Technology), first this is the evolution of the stock and the **different actions (buy / sell / hold)** to perform every day.

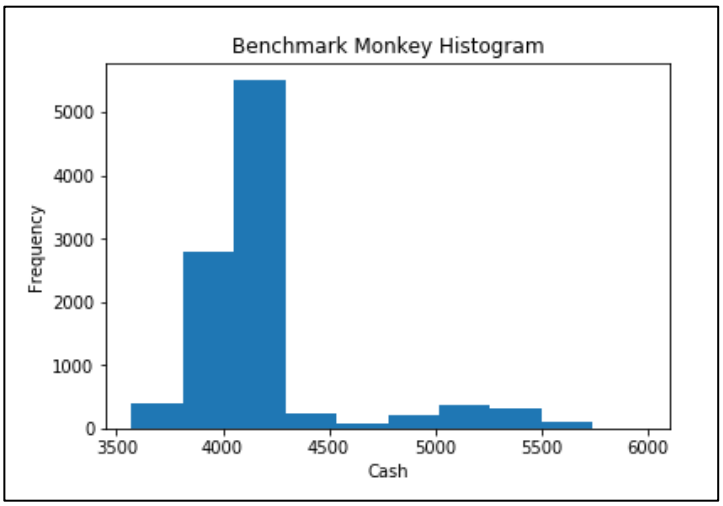


To see how much, we have earned for the last year, we have plotted the Profit and Loss and the Cash chart.





For each stock, we have compared it to a naïve algorithm (we call it benchmark monkey) which acts like a monkey and **chooses randomly the action to perform every day**, we have run this algorithm 10 000 times in order to get a histogram to compare with:



Finally, reinforcement learning is a vast field of research very active and unfortunately we did not have the time to **implement the latest results about those methods such as the "Policy Gradient", "Actor-Critic" or also the evaluation of tweet** related to the company thanks to deep learning analysis.

However, we have coded and realized a **trading system based on the Q-learning algorithm** (as well as its improved variants) and which obtained better results than the naïve algorithm in a large majority of stocks (80 / 100). Finally, we could not refine our algorithm to beat more sophisticated algorithms because we were running out of time and **running the program required an average of 16 hours...**

In fact a lot of companies and **data scientist are now using these techniques** which can be applied to many fields such as automatic cars, image recognition, financial markets and so on…

X. Future Improvements

A- Policy Gradient

With theta, alpha and beta the weight of the layers used to approximate the functions.
In fact the Deep Q learning is not always accurate because the neural network is only an approximation of the Q function so sometimes it leads to a local minimum instead of a global one.
To solve this problem, we are now focusing of a new kind of reinforcement learning method called the "Policy Gradient".

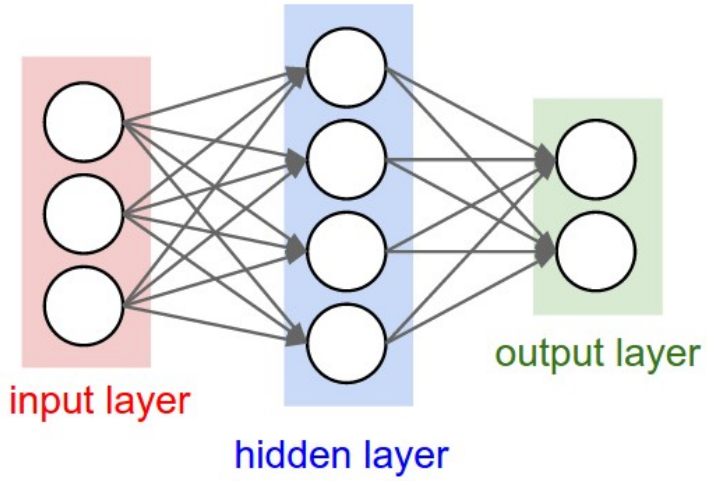
Definition: policy

A policy is a probability to take an action given a certain state, we denote the policy as π.

$$\pi = P(a\,|s)$$

In the case of the "Policy Gradient", we are not trying to follow the Q function policy, we are now looking for the best policy to perform and update.

The neural network that we are going to use is kind of similar, as inputs there are close values of the previous days and as outputs there is the set of 3 actions: buy, hold and sell.



Every day we obtain a different set probabilities set of the 3 actions, we denote it as a vector:

$$\pi = \begin{pmatrix} 0.63 \\ 0.4 \\ 0.1 \end{pmatrix}$$

The action, which has the highest probability, is going to be performed. The question is was it the best action to do? And how can we update and adjust the neural network in order to finally get the better results.

Firstly, we write the weight of the neural network as θ, we have to adjust and modify this vector of parameters in order to obtain a policy $\pi(a, s, \theta)$ which gives the best results. This is an optimization problem, and the function to optimize is the following:

$$J(\theta) = E[\pi(a, s, \theta) * r(a, s)]$$

Where r is the reward,

It means that we are looking for the best expectancy of the reward obtained following the policy vector. To make it clear, we are looking for the direction to update θ in order to maximize the future rewards. The solution is to perform a gradient descent upon the optimization function in order to find the best direction and update it accordingly:

$$\theta' = \theta + \alpha * \nabla\big(J(\theta)\big)$$

There are many possibilities to compute the gradient, for example we can use the finite difference method…
In our case, we are going to perform the same method as it is used by DeepMind: it is based on the **Policy Gradient Theorem** (Sutton, 1999) where we can find an easy way to compute the gradient descent.

$$\nabla\big(J(\theta)\big) = E[\, \nabla\, log(\, \pi(a, s, \theta)\,) * r(a, s)\,]$$

To make it simple and clear:

- We take the vector of policies π (i.e the outputs of the network)
- We take the log of it, we perform the gradient
- Then we multiply it with the reward that we have obtained
- We take the expectation of the all, i.e. the mean
- Then we can change the weight of the neural network into the best direction

If we run it a lot of times (epochs) then the **network will convey to the optimal solution** and after that it will know **how to act best at each state**.

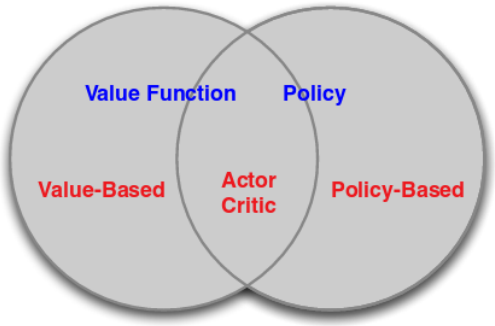
B- Actor-Critic Estimation

Nowadays, firms and researchers specialized in reinforcement learning no longer use only policy gradient or Deep Q Learning but they use a mix of both which is called the "Actor-Critic" strategy.

With that strategy, two neural networks are now used:

-the first one is the "actor network" it is dedicated to finding the best action to perform at each state
-the other network is called the "critic network", his role is to evaluate the strategy of the actor network and estimates if this action was a good one or not. It is similar to the Q-Learning estimation value

By combining both networks and strategies, it is possible to obtain better performances and with a set of actions and states more important.



C- Twitter Analysis with Deep Learning

We can also use Deep Learning combined with sentiments analysis in order to get the feeelings about a firm on Twitter because their correlation has been shown by **Venkata Sasank Pagolu**(2016) :
"It is shown that a strong correlation exists between the rise and fall in stock prices with the public sentiments in tweets."

Thanks to this approach, we will use an indicator of the emotion to predict evolution of stock markets and make our trading algorithm more efficient.

The first thing that is to be done is to train a neural network in order to detect and recognize the feelings given a sentence. As inputs there is a simple sentence issued from a tweet just as "The firm X has dismissed 1000 workers" and as outputs the neural network has to be able to recognize if this sentence was confident, mistrustful or just neutral.

Once the neural network is trained, we are using a library called TextBlob which helps us to gather all the tweets concerning a specific company on a day and tells us what the general impression about that company on that day is. Then, we use this feeling as another indicator (in addition to the close values, moving average…) and feed the actor neural network with it.

XI. References

Bernal, A., Fok, S., & Pidaparthi, R. (2012). Financial Market Time Series Prediction with Recurrent Neural Networks.

[Bollen et al., 2011] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. Journal of Computational Science, 2(1):1–8, 2011.

[Bordes et al., 2011] Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. Learning structured embeddings of knowledge bases. In Proc. of AAAI, 2011.

Breiman, 2001L. Breiman, Random forestsMachine Learning, 45 (1) (2001), pp. 5–32

[Cutler et al., 1998] David M Cutler, James M Poterba, and Lawrence H Summers. What moves stock prices? Bernstein,Peter L. and Frank L. Fabozzi, pages 56–63, 1998.

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). Time series analysis: forecasting and control. John Wiley & Sons.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148(34), 13.

Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach (Vol. 5). GMD-Forschungszentrum Informationstechnik.

Luss and d'Aspremont, 2012 Ronny Luss and Alexandre d'Aspremont. Predicting abnormal returns from news using text classification. Quantitative Finance,(doi:10.1080/14697688.2012.672762):1–14, 2012.

M. Ghil, M.R. Allen, M.D. Dettinger, K. Ide, D. Kondrashov, M.E. Mann, A.W. Robertson,

Saunders, Y. Tian, and F. Varadi. Advanced spectral methods for climatic time series. Reviews of geophysics, 40(1) :3–1, 2002.

N. Golyandina, V. Nekrutkin, and A.A. Zhigljavsky , Analysis of time series structure :SSA and related techniques. CRC Press, 2010.

Allan Timmermann*, Clive W.J 2004. , Efficient market hypothesis and forecasting ,Granger Department of Economics, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0508, USA

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural computation, 14(11), 2531-2560.

Roland Gillet & Ariane Szafarz, 2004."L'efficience informationnelle des marchés: une hypothèse, et au-delà ?,"Working Papers CEB 04-004.RS, ULB -- Universite Libre de Bruxelles.

Deep Q-trading(2016)Yang Wang1,3*, Dong Wang1,2, Shiyue Zhang1,5, Yang Feng1,4, Shiyao Li1,5 and Qiang Zhou1,2