ENSIIE

Research project report

Option pricing and implied volatility computation using a data-driven approach

Imane Alla

Mame Diarra Toure

Enseignant :

M. Pulido

14/05/2020

5 juin 2020

Table des matières

1 Introduction

In computational finance, numerical methods are commonly used for the valuation of financial derivatives. The applicability of any given option pricing method (Fourier pricing, PDE methods, asymptotic methods, Monte Carlo, ...etc.) depends on the regularity properties of the particular stochastic model at hand. Therefore, tractability of stochastic models has been one of the most decisive qualities in determining their popularity. In fact it is often a more important quality than the modelling accuracy itself[1]. There are more realistic and more accurate stochastic market models than the popular one that are used nowadays but the main issue remain in the efficiency of the computation. Often high quality asset models are discarded due to a slow calibration time.
The price of an option is linked to variables such a the volatility(σ), the interest rate(r), the time to maturity(T), the strike (K) ect... by a complex non linear function
In this paper we want to use machine learning algorithms in order to approximate that function. Starting with basic algorithms such as polynomial regression , we are going to focus mainly on the use of neural networks. Artificial neural networks (ANNs) with multiple hidden layers have become successful machine learning methods to extract features and detect patterns from a large data set. There are different neural network variants for particular tasks, for example, convolutions neural networks for image recognition and recurrent neural networks for time series analysis. It is well-known that ANNs can approximate nonlinear functions and can thus be used to approximate solutions to PDEs[2].

2 Option pricing : The Heston Model

2.1 Presentation

Despite its tremendous success, the Black-Scholes model of option pricing has some well-known deficiencies, perhaps the most important of which is the assumption that **the volatility of the return on the underlying asset is constant**[3]. The question then arises as to how to price options in a way which is consistent with the market-observed variation of implied volatility. Although it is easy to modify the Black-Scholes model to take into account of the term structure of implied volatilities, the variation

with strike price(volatility smile) cannot be incorporated into the Black-Scholes model. One of the concepts used to cope with this problem is that of stochastic volatility. The constant volatility of the Black-Scholes model corresponds to the assumption that the underlying asset follows a lognormal stochastic process. The basic assumption of stochastic volatility models is that the of the underlying asset is itself a random variable. There are two Brownian motions : one for the underlying, and one for the variance; stochastic volatility model are thus two-factor models. Of course, the two processes are correlated and, at least in the equity world, the correlation is usually taken to be negative : increases/decreases in the asset price tend to be coupled to decreases/increases in the volatility.Once the variance of the underlying has been made stochastic, closed-form solutions for European call and put options will in general no longer exist. One of the attractive features of the Heston model, however, is that closed-form solutions do exist for European plain vanilla options.
Heston (1993) proposed the following model :[4]

$$dS_t = \mu S_t dt + \sqrt{V_t} S_t dW_t^1 \tag{1}$$

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{(V_t)}dW_t^2 \tag{2}$$

$$dW_t^1 dW_t^2 = \rho dt \tag{3}$$

Where $\{S_t\}_{t\geq 0}$ and $\{V_t\}_{t\geq 0}$ are the price and volatility processes, respectively, and $\{W_t^1\}_{t\geq 0}$, $\{W_t^2\}_{t\geq 0}$ are correlated Brownian motion processes (with correlation parameter ρ). $\{V_t\}_{t\geq 0}$ is a square root mean reverting process, with long-run mean θ, and rate of reversion κ. σ referred to as the volatility of volatility. All the parameters$(\mu, \kappa, \theta, \sigma, \rho)$, are time and state homogenous.

2.1.1 Pricing Options

In the Black-Scholes model, a contingent claim is dependent on one or more tradable assets. The randomness in the option value is solely due to the randomness of these assets. Since the assets are tradable, the option can be hedged by continuously trading the underlyings. This makes the market complete, i.e.,every contingent claim can be replicated.In a stochastic volatility model, a contingent

claim is dependent on the randomness of the asset ($\{S_t\}_{t\geq 0}$) and the randomness associated with the volatility of the asset's return ($\{V_t\}_{t\geq 0}$). Only one of these is tradable(the asset).Volatility is not a traded asset. This renders the market incomplete and has many implications to the pricing of options.

2.1.2 PDE approach

Under the Heston model, the value of any option,$U(S_t, V_t, t, T)$, must satisfy the following partial differential equation :

$$\frac{1}{2}VS^2\frac{\partial^2 U}{\partial S^2} + \rho\sigma VS\frac{\partial^2 U}{\partial S\partial V} + \frac{1}{2}\sigma^2 V\frac{\partial^2 U}{\partial V^2} + rS\frac{\partial U}{\partial S} + \{\kappa(\theta - V) - \Lambda(S, v, t)\sigma\sqrt{V}\}\frac{\partial U}{\partial V} - rU + \frac{\partial U}{\partial t} = 0 \quad (4)$$

$\Lambda(S, v, t)$ is called the market price of volatility risk. Heston assumes that the market price of volatility risk is proportional to volatility, i.e.

$$\begin{cases} \Lambda(S, v, t) = k\sqrt{V} \\ \Lambda(S, v, t)\sigma\sqrt{V} = k\sigma V_t = \lambda(S, V, t) \end{cases} \quad (5)$$

Where $\lambda(S, V, t)$ therefore represents the market price of volatility risk.

2.1.3 Risk Neutral approach

Risk neutral valuation is the pricing of a contingent claim in an equivalent martingale measure (EMM). The price is evaluated as the expected discounted payoff of the contingent claim, under the EMM $\mathbb{Q}$:

$$Option Value = \mathbb{E}_t^{\mathbb{Q}}[e^{r(T-t)}H(T)] \quad (6)$$

where H(T) is the payoff of the option at time T and r is the risk free rate of interest over [t, T] (Interest rates are deterministic and previsible, and that the numeraire is the money market instrument). Note that moving from a real world measure to an EMM is achieved by Girsavov's Theorem.

2.2 Computational Evaluation : Fourier Transform

In order to practically implement such a model, one needs to be able to compute the price of vanilla options implied by it.This section, therefore, provides the tools required to perform such calculations.

2.2.1 Definition

The Fourier Transform $F(\{.\})$ and inverse Fourier transform $F^{-1}(\{.\})$ of and integrable function,f(x) is :

$$F(f(x)) = \int_{-\infty}^{\infty} e^{iux} f(x)\,dx = \phi_X(u) \tag{7}$$

Where i being the imaginary unit, and u a real number. In the case when f(x) = $f_X(x)$ is a probability density function of a random variable X, the transform above is the characteristic function of X to be denoted by $\phi_X(u)$.

$$F^{-1}(\phi_X(u)) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-iux}\phi_X(u)\,du = f_X(x) \tag{8}$$

For existence of the inverse transform,$\phi_X(u)$ has to fulfill the integrability condition.

2.3 Fourier Transform

The famous risk-neutral option pricing approach, employing the Feynman-Kac theorem, involves taking expectation of the contract's payoff at maturity with respect to a risk neutral measure and thus the relevant probability density function (PDF) of the random log-price is needed to perform the calculation. In practice, a PDF is not always easily obtained. Instead, characteristic functions always exist. The general idea behind using the techniques upper is that by using characteristic functions, one is being transferred to making computations within the Image space rather than the Real one. It turns out that for a number of functions the former operations are much easier. That is, for more complicated processes, one can employ the relationship between density and characteristic function of a distribution, with the procedure being composed in two steps. First, a mapping of the characteristic function to the payoff of interest is generated and second, the inversion formula is applied to obtain the prices of interest. A technique involving Fourier transform methods was used by Heston (1993) for

obtaining the widely used pricing formulae. This method has been proposed by (Carr Madan 1999). The method of **Carr and Madan** consists of calculating the Fourier transform of the entire option price without separating the payoff from the risk-neutral density. We will discuss the case of a **call option**.The approach begins with the familiar risk-neutral valuation, which for a European call price with a strike K and a spot at maturity ST satisfies the relation :

$$C_T(k) = e^{-rT}\mathbb{E}^{\mathbb{Q}}[(S_T - K)^+] = e^{-rT}\int_k^{\infty}(e^x - e^k)q(x)dx \tag{9}$$

where x = log S , k =log k and q(x) is the risk-neutral density. The main idea is to transform the expression for $C_T(k)$ and then use the inversion formula and thus obtain the desired price. In that way the unknown density function q(x) is avoided.

However, due to the fact that $\lim_{k\to+\infty} C_T(k)$ = S0, the condition for absolute integrabilityis not fulfilled and this means that the Fourier transform of C_T does not exist. To avoid this problem,we modify the expression for the call price C_T , by introducing a damping factor of the forme exp αk with α> 0. Then, the new function :

$$c_T(k) = e^{\alpha k}C_T(k) \tag{10}$$

is absolutely integrable for a suitably chosen α and hence both the Fourier transform and the inverse transform exist and as k approaches minus infinity the price of a call goes to zero, as it should.

Let $\Psi(u)$ denote the Fourier transform of $c_T(k)$. Then it takes the form :

$$\Psi(u) = \frac{\phi(u-(\alpha+1)i)}{(\alpha+iu)(\alpha+1+iu)} \tag{11}$$

with $\phi(u)$ being the characteristic function of the price process under the risk-neutral setting s.t :

$$\phi(u) = \mathbb{E}[\exp(iu\ln(S_t))] \tag{12}$$

$$= \frac{\exp(iu\ln(S_0) + iurt + \frac{\kappa\theta(\kappa - i\rho\sigma u)}{\sigma^2})}{(\cosh(\frac{\gamma t}{2}) + \frac{\kappa - i\rho\sigma u}{\gamma}\sinh(\frac{\gamma t}{2}))^{\frac{2\kappa\theta}{\sigma^2}}}\exp\left(\frac{-(u^2+iu)V_0}{\gamma\coth(\frac{\gamma t}{2}) + \kappa - i\sigma\rho u}\right) \tag{13}$$

Where $\gamma = \sqrt{\sigma^2(u^2 + iu) + (\kappa - i\rho\sigma u)^2}$,r is the risk free rate and,S_0 and V_0 are the initial values of the price process and the volatility process.

Finally, the call price is obtained by taking the inverse transform :

$$C_T(k) = \frac{\exp(-\alpha k - rT)}{2\pi} \int_{-\infty}^{\infty} \exp(-iuk)\Psi(u)du \tag{14}$$

$$= \frac{\exp(-\alpha k - rT)}{2\pi} \int_{0}^{\infty} Re[\exp(-iuk)\Psi(u)]du \tag{15}$$

3 Implied volatility : The Black Scholes model

3.1 presentation

The Black Scholes model, also known as the Black-Scholes-Merton (BSM) model, is a mathematical model for pricing an options contract. In particular, the model estimates the variation over time of financial instruments. It assumes these instruments (such as stocks or futures) will have a lognormal distribution of prices. Using this assumption and factoring in other important variables, the equation derives the price of a call option. The model assumes the price of heavily traded assets follows a geometric Brownian motion with constant drift and volatility. When applied to a stock option, the model incorporates the constant price variation of the stock, the time value of money, the option's strike price, and the time to the option's expiry. We denote S_t the price of a non dividend paying asset. it follows a geometrical brownian motion dynamic :

$$dS_t = \mu S_t dt + \sigma S_t dBt \tag{16}$$

where B_t is a standard brownian motion, with t being the time, μ the drift parameter, and σ the volatility of the asset. The model is valid on the interval [0,T], where T is the maturity of tThe option. That stochastic differential equation has a closed form solution :

$$S_t = S_0 \exp(\mu t - \frac{\sigma^2}{2} t + \sigma B_t) \tag{17}$$

where S_0 is the spot price observed at time 0. One particular result from this model is that the law of S_t is lognormal (i.e. its logarithm follows a normal law).

3.2 Some information about Strategies

A strategy will be define as a process $\phi = (\phi_t)_{0\leq t\leq T} = (H_t^0, H_t)$ with value in $\mathbb{R}^2$, adapted to the natural filtration F_t of the brownian motion. the component H_t^0 and H_t are the quantities of risk less asset and risky asset respectively held in the portfolio at time t. The value of the portfolio at time t is given by :

$$V_t(\phi) = H_t^0 S_t^0 + H_t S_t \tag{18}$$

We say that a strategy is self-financing if it satisfies the following condition

1. $\int_0^T |H_t^0| dt + \int_0^T H_t^2 | dt < +\infty$ almost surely
2. $H_t^0 S_t^0 + H_t S_t = H_0^0 S_0^0 + H_0 S_0 + \int_0^t H_u^0 dS_u^0 + \int_0^t H_u dS_u$

If we denote $\tilde{S_t} = \exp(-rt)S_t$ the discounted value of the asset we then can say that the strategy ϕ is self financing if and only if :

$$\tilde{V}_t(\phi) = V_0(\phi) + \int_0^t H_u d\tilde{S}_u \quad a.s \tag{19}$$

We say that a strategy ϕ is admissible if it is self-financing and if the discounted value $\tilde{V}_t = H_t^0 + H_t \tilde{S}_t$ of the corresponding portfolio is for all t non-negative and such that the $sup_{t\in[0,T]} \tilde{V}_t$ is square integrable under the probability $\mathbb{P}^*$ (the probability such as the discounted price of the asset is a martingale).

3.3 The "risque neutre" probability

We consider the model :

$$dS_t = \mu S_t dt + \sigma S_t dBt \tag{20}$$

using that SDE and applying the itô's formula to the discounted asset price $\tilde{S}_t = \exp(-rt)S_t$ we have :

$$d\tilde{S}_t = -r\exp(-rt)S_t dt + \exp(-rt)dS_t \tag{21}$$

$$d\tilde{S}_t = \tilde{S}_t((\mu - r)dt + \sigma dB_t) \tag{22}$$

Consequently if we set $W_t = B_t + \frac{(\mu-r)t}{\sigma}$ then we have :

$$d\tilde{S}_t = \tilde{S}_t \sigma dW_t \tag{23}$$

and then by the grisanov theorem we have there exist a probability $mathbbP^*$ under which W_t is a standard brownian motion and thus $\tilde{S}_t = \tilde{S}_0 \exp(\sigma W_t - \frac{\sigma^2 t}{2})$ [5]

3.4 Pricing

In this section we will focus on European options. An European option will be defined by a non-negative $F_T - measurable$ random variable h. Our study focus on call option with h defined by $f(S_t) = (S_t - K)^+$ An option is said to be repliacable if its payoff at maturity is equal to the final value of an admissible strategy. It is clear that for the option defined by h to be replicable, it is necessay that h should be square integrable under $\mathbb{P}^*$ which is the case for a call. By the mean of AOA to price our option we're gonna use a repliacting startegy which we know does exist in the black scholes model .
Suppose that there is an admissible strategy (H^0, H) replicating the option the value at time time t of the replicating portfolio is $V_t(\phi) = H_t^0 S_t^0 + H_t S_t$ and by assumption we have $V_T = h/$
Let $\tilde{V}_t = V_t \exp(-rt)$ be the discounted value then $\tilde{V}_t = H_t^0 + H_t \tilde{S}_t$ since the strategy is self financing we have :

$$\tilde{V}_t(\phi) = V_0(\phi) + \int_0^t H_u d\tilde{S}_u \tilde{V}_t(\phi) \qquad = V_0(\phi) + \int_0^t H_u \sigma \tilde{S}_u dW_u. \tag{24}$$

Remember that under $\mathbb{P}^*$, $sup_{t\in[0,T]}\tilde{V}_t$ is square integrable by definition of admissible strategies. Furthermore, the precedeing equality shows that the process $\tilde{V}_t$ is a square integrable martingale under

$\mathbb{P}^*$.

Hence $\tilde{V}_t = \mathbb{E}^*[\tilde{V}_T|F_t]$ and consequently $V_t = \mathbb{E}^*[\exp(-r(T-t))h|F_t]$

Since we know for a call that $h = f(S_T) = (S_T - K)^+$ we can expresse the value of the option V_t at time t as a function of t and of the asset price S_t. We have :

$$V_t = \mathbb{E}^*[\exp(-r(T-t))f(S_T)|F_t] \tag{25}$$

We deduce then that $S_T = S_t \exp(r(T-t))\exp(\sigma(W_T - W - t)) - (\frac{\sigma}{22(T-t))}$ hence

$$V_t = \mathbb{E}^*[\exp(-r(T-t))f(S_t \exp(r(T-t))\exp(\sigma(W_T - W - t)) - (\frac{\sigma^2}{2}(T-t)))|F_t]$$

. The random variable S_t is $F_t - measurable$ and under $\mathbb{P}^*$, $W_T - W_t$ is independant of F_t therefore we can write

$$V_t = \mathbb{E}^*[\exp(-r(T-t))f(S_t \exp(r(T-t))\exp(\sigma(W_T - W_t)) - (\frac{\sigma^2}{2}(T-t)))]$$

Let denote

$$F(t,x) = \mathbb{E}^*[\exp(-r(T-t))f(x\exp(r(T_t))\exp(\sigma(W_T - W - t)) - (\frac{\sigma^2}{2}(T-t)))]$$

Since under $\mathbb{P}^*$ we know that $W_T - W_t$ is a zero-mean normal variable with vairance (T-t) we can write

$$F(t,x) = \exp(-r(T-t))\int_{-\infty}^{+\infty} f(x\exp((r - \frac{\sigma^2}{2})(T-t)) + \sigma y\sqrt{T-t})\exp(\frac{-y^2/2)dy}{\sqrt{2\pi}}$$

for a call $f(S_t) = (S_t - K)^+$

$$F(t,x) = \mathbb{E}^*[\exp(-r(T-t))(x\exp((r - \frac{\sigma^2}{2})(T-t) + (\sigma(W_T - W - t))) - K)^+]$$

$$F(t,x) = \mathbb{E}[x\exp(\sigma g - \sigma^2\theta/2 - K\exp(-r\theta))^+]$$

where g is a Gaussian variable and $\theta = (T - t)$ We set

$$d_1 = log(x/K) + (r + \sigma^2/2)\theta\sigma\sqrt{\theta}$$

and

$$d_2 = d1 - \sigma\sqrt{\theta}$$

using these notation we have

$$F(t,x) = \mathbb{E}[(x\exp(\sigma g - \sigma^2\theta/2 - K\exp(-r\theta)))\mathbb{1}_{g+d_2\geq 0}]$$

$$F(t,x) = \int_{-d_2}^{+\infty}((x\exp(\sigma g - \sigma^2\theta/2 - K\exp(-r\theta))))exp(\frac{-y^2/2)dy}{\sqrt{2\pi}}$$

Writing this expression as the difference of two integrals and using the change of variables $z = y + sigma\sqrt{\theta}$ we have

$$F(t,x) = xN(d_1) - K\exp(-r\theta)N(d_2)$$

where N is the cumulative distribution function of a Gaussian random variable .
Finally we can write $V_c(t,S)$ the value of an european call option on the underlying asset at time t :

$$V_c(t,S) = SN(d_1) - K\exp(-r\theta)N(d_2)[5] \tag{26}$$

3.5 implied volatility

Implied volatility is considered an important quantity in finance. Given an observed market option price $V^m kt$, the Black-Scholes implied volatility σ can be determined by solving $BS(\sigma^{;S,K,\theta,r)=V^m kt}$ The monotonicity of the Black-Scholes equation with respect to the volatility guarantees the existence of $\sigma^{\epsilon[0,+\infty]}$. We can write the implied volatility as an implicit formula, $\sigma^*(K,T) = BS^{-1}(V^m kt, S, K, \theta, r)$ where BS^{-1} denotes the inverse black scholes function.[2] The value of σ^* is determined by means of a numerical iterative technique like the newton-raphson algorithm or the brent algorithm , since the previous equation can be converted into a root-finding problem.

For a given option price ($V^{m}kt$), interest rate (r), and initial asset price S_0 we used the newton raphson algorithm to compute the implied volatility as a function of the maturity T and the strike K. We have the following surface plot

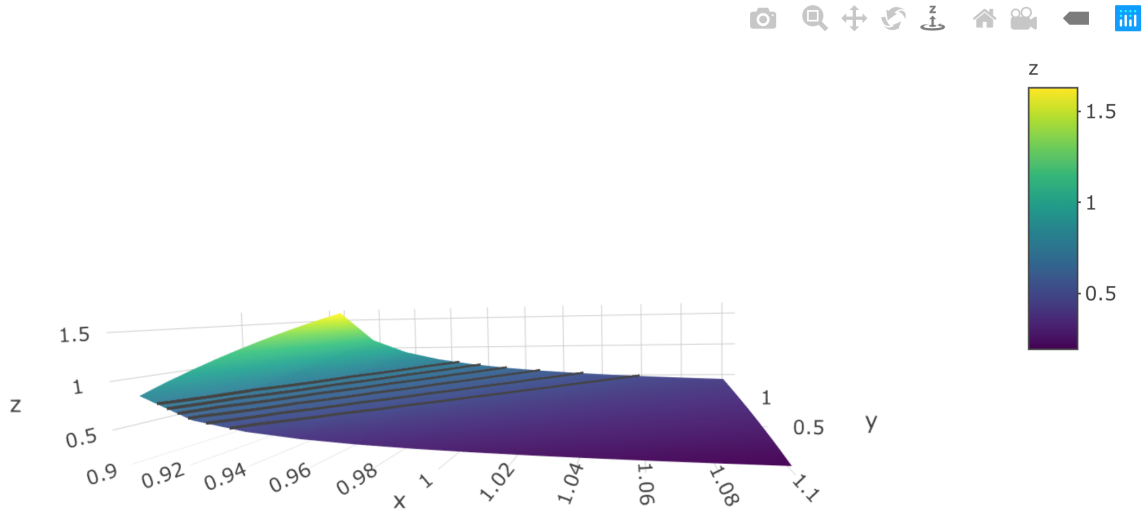

FIGURE 1 – Surface of implied volatility(z-axis) according to the maturity T(y-axis) and the strikes K (x-axis)

In the next part we're going to introduce a data driven approach in order to compute the implied volatility

4 The ML algorithm we are going to use and how they work

4.1 Ordinary Least square method(OLS)

In statistics, ordinary least squares (OLS) is a type of linear least squares method for estimating the unknown parameters in a linear regression model. OLS chooses the parameters of a linear function of

a set of explanatory variables by the principle of least squares : minimizing the sum of the squares of the differences between the observed dependent variable (values of the variable being observed) in the given dataset and those predicted by the linear function. Consider an overdetermined system :

$$\sum_{j=1}^{p} \mathrm{X}_{ij}\beta_j = y_i, \quad (i = 1, 2, \ldots, n) \tag{27}$$

of n linear equations in p unknown coefficients, 1,2,...,p, with n > p.

Note : for a linear model as above, not all of X contains information on the data points. The first column is populated with ones,$\mathrm{X}_{i1} = 1$, only the other columns contain actual data, so here p = number of regressors + 1. This can be written in matrix form as :

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{y} \tag{28}$$

Where :

$$\mathbf{X} = \begin{pmatrix} \mathrm{X}_{11} & \mathrm{X}_{12} & \cdots & \mathrm{X}_{1p} \\ \mathrm{X}_{21} & \mathrm{X}_{22} & \cdots & \mathrm{X}_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{X}_{n1} & \mathrm{X}_{n2} & \cdots & \mathrm{X}_{np}, \end{pmatrix} \quad \boldsymbol{\beta} = {}^{\mathrm{T}}(\beta_1 \beta_2 \ldots \beta_p), \qquad \mathbf{y} = {}^{\mathrm{T}}(y_1 y_2 \ldots y_n) \tag{29}$$

Such a system usually has no exact solution, so the goal is instead to find the coefficients $\boldsymbol{\beta}$ which fit the equations "best", in the sense of solving the quadratic minimization problem

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\arg\min}\, \mathrm{S}(\boldsymbol{\beta}) \tag{30}$$

where the objective function S is given by :

$$\mathrm{S}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left| y_i - \sum_{j=1}^{p} \mathrm{X}_{ij}\beta_j \right|^2 = \left\| \mathbf{y} - \mathbf{X}\boldsymbol{\beta} \right\|^2. \tag{31}$$

This minimization problem has a unique solution, provided that the p columns of the matrix X are linearly independent, given by solving the normal equations

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})\hat{\boldsymbol{\beta}} = \mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{32}$$

4.2 Penalized Regression

4.2.1 Ridge Regression

Ridge regression is a parsimonious model which performs L2 regularization. The L2 regularization adds a penality equivalent to the square of the maginitude of regression coefficients and tries to minimize them. In Ridge Regression, the OLS loss function is augmented in such a way that we not only minimize the sum of squared residuals but also penalize the size of parameter estimates, in order to shrink them towards zero :

$$L_{ridge}(\hat{\beta}) = \sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j-1}^{m}\hat{\beta}_j^2 = ||y - X\hat{\beta}||^2 + \lambda||\hat{\beta}||^2.$$

Ridge regression does a proportional shrinkage and handles collinear variables but it does not perform a selection.We use the GridSearchCVfunctione to find the best λ. This function does k-fold cross-validation and returns a value for the best lambda value which is equal in our case to $\lambda_{ridge} = 0.01$.

4.2.2 Lasso

Lasso, or Least Absolute Shrinkage and Selection Operator, is quite similar conceptually to ridge regression. It also adds a penalty for non-zero coefficients, but unlike ridge regression which penalizes sum of squared coefficients (the so-called L2 penalty), lasso penalizes the sum of their absolute values (L1 penalty). As a result, for high values of , many coefficients are exactly zeroed under lasso, which is never the case in ridge regression. This is called "soft thresholding.

The only difference in ridge and lasso loss functions is in the penalty terms. Under lasso, the loss is defined as :

$$L_{lasso}(\beta) = \sum_{i=1}^{n}(y_i - x_i\beta)^2 + \lambda \sum_{i=1}^{m}|\beta_i| \tag{33}$$

4.2.3 k-nearest neighbor(KNN)

k-Nearest Neighbors (k-NN) is an algorithm that is useful for making classifications/predictions when there are potential non-linear boundaries separating classes or values of interest. Conceptually, k-NN examines the classes/values of the points around it (i.e., its neighbors) to determine the value of the point of interest. The majority or average value will be assigned to the point of interest. A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the same distance functions as KNN classification.

4.3 Polynomial Regression

In statistics, polynomial regression is a form of regression analysis in which the relationship between the independent variables X and the target y is modelled as an nth degree polynomial in X. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x). Although polynomial regression fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E(y | x) is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is considered to be a special case of multiple linear regression. The polynomial regression model $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_m x_i^m + \varepsilon_i$ $(i = 1, 2, \ldots, n)$ can be expressed in matrix form in terms of a design matrix **X**, a response vector $\vec{y}$, a parameter vector $\vec{\beta}$, $\vec{\varepsilon}$ of random errors. The i-th row of

X and $\vec{y}$ will contain the x and y value for the i-th data sample. Then the model can be written as a system of linear equations :

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix},$$

which when using pure matrix notation is written as

$\vec{y} = \mathbf{X}\vec{\beta} + \vec{\varepsilon}$. The vector of estimated polynomial regression coefficients (using ordinary least squares estimation) is

$\hat{\vec{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\,\mathbf{X}^{\mathrm{T}}\vec{y}$, The explanatory (independent) variables resulting from the polynomial expansion of the "baseline" variables are known as higher-degree terms. Such variables are also used in classification setting.

4.4 Artificial Neural Networks

ANNs generally constitute three levels of components, i.e. neurons, layers and the architecture from bottom to top. The architecture is determined by a combination of different layers, that are made up of numerous artificial neurons. A neuron, which involves learnable weights and biases, is the fundamental unit of ANNs. By connecting the neurons of adjacent layers, output signals of a previous layer enter a next layer as input signal. By stacking layers on top of each other, signals travel from the input layer through the hidden layers to the output layer potentially through cyclic or recurrent connections, and the ANN builds a mapping among input-output pairs.[2] As shown in the next figure an artificial neuron basically consists of the following three consecutive operations :

1. Calculation of a summation of weighted inputs,
2. Addition of a bias to the summation,

3. Computation of the output by means of a transfer function

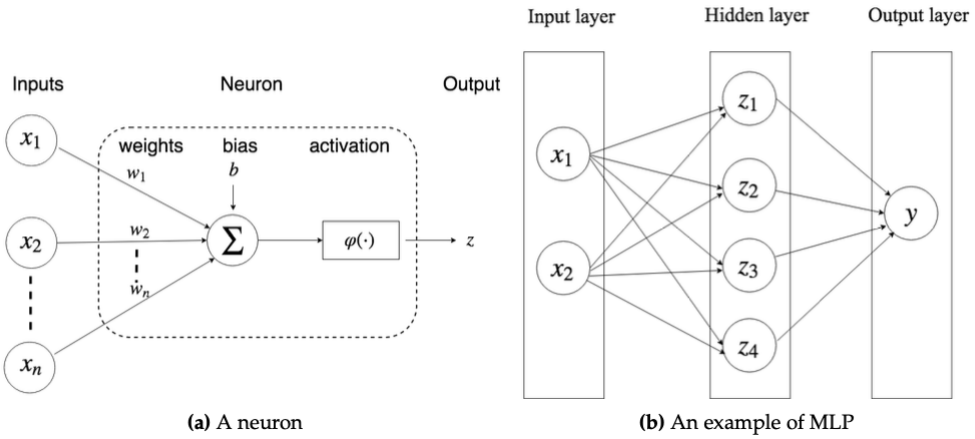

FIGURE 2 – Representation of an ANN

A multi-layer perceptron (MLP) consisting of at least three layers is the simplest version of an ANN. Mathematically, MLP's are defined by the following parameters

$$\theta = (W_1, b_1, ..., W_L, b_L)$$

where W_j is a weighed matrix and b_j is the bias vector of the L-th neural layer. A function can thn be expressed as follows

$$y(x) = F(x|\theta)$$

According to the Universal Approximation Theorem , a single-hidden-layer ANN with a sufficient number of neurons can approximate any continuous function. Hence we can use ANN to approximate the relation between the implied volatility and the explantory varibales.

4.5 XGboost

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

When using gradient boosting for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs that contains a continuous score. XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions). The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It's called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

5 Statiscal method for Option pricing

5.1 Visualization

5.1.1 Generating the Database

We were challenged to build a model that approximate European call option prices. First we generate our data using the previous model(Heston model).We implemented the pricer of the heston model using the Fourier formula and used that function to create the dataset with the parameters being the explanatory vairables and the price the target. The data set composed of 10 000 observations of 10 initial variables with the target variable being the **Price**. First, we will start by cleaning up our data set. That means getting rid of missing values and any observation that seem like an outlier. Secondly, we will look at the different relation between the feature variables and the target variable. Thirdly we are going to compute various machine learning algorithms going from a simple linear model to more sophisticated algorithms such as neural network algorithms. Finally, we will compute the boosting algorithm and see if it improve our predictions.

	K	S0	V0	r	kappa	theta	sigma	rho	T	Price
0	1.115590	1	0.366737	0.060365	1.671733	0.157764	0.320192	-0.148896	0.180108	0.059049
1	1.544502	1	0.299991	0.044878	0.012699	0.388983	0.328124	-0.427174	1.249526	0.099795
2	0.728111	1	0.462448	0.048523	1.276814	0.209936	0.371584	-0.527093	0.806844	0.370892
3	1.461709	1	0.276264	0.033629	0.277704	0.126179	0.061917	-0.678341	1.387362	0.117079
4	1.155376	1	0.365953	0.042299	0.454751	0.250158	0.399999	-0.200558	0.426793	0.103244
...	...	...	...	...	...	...	...	...	...	...
9562	1.483364	1	0.169162	0.034599	0.805419	0.284269	0.483452	-0.506029	1.368025	0.077133
9563	0.739450	1	0.217119	0.015998	0.828347	0.186141	0.115298	-0.626611	0.448161	0.288783
9564	1.161337	1	0.427014	0.029945	1.226045	0.354683	0.215935	-0.260698	0.154996	0.048013
9565	0.900616	1	0.230047	0.090707	1.489304	0.409354	0.337963	-0.922097	0.723084	0.259687
9566	1.229842	1	0.428947	0.060666	0.827646	0.222676	0.335839	-0.333283	0.397540	0.087869

9567 rows × 10 columns

FIGURE 3 – Data set

5.1.2 Correlation plot

On the correlation plot of the remaining variables, we can see that the target variable(Price) is strongly correlated with the Strike (K) and the maturity (T). Note that we chose $S_0 = 1$.

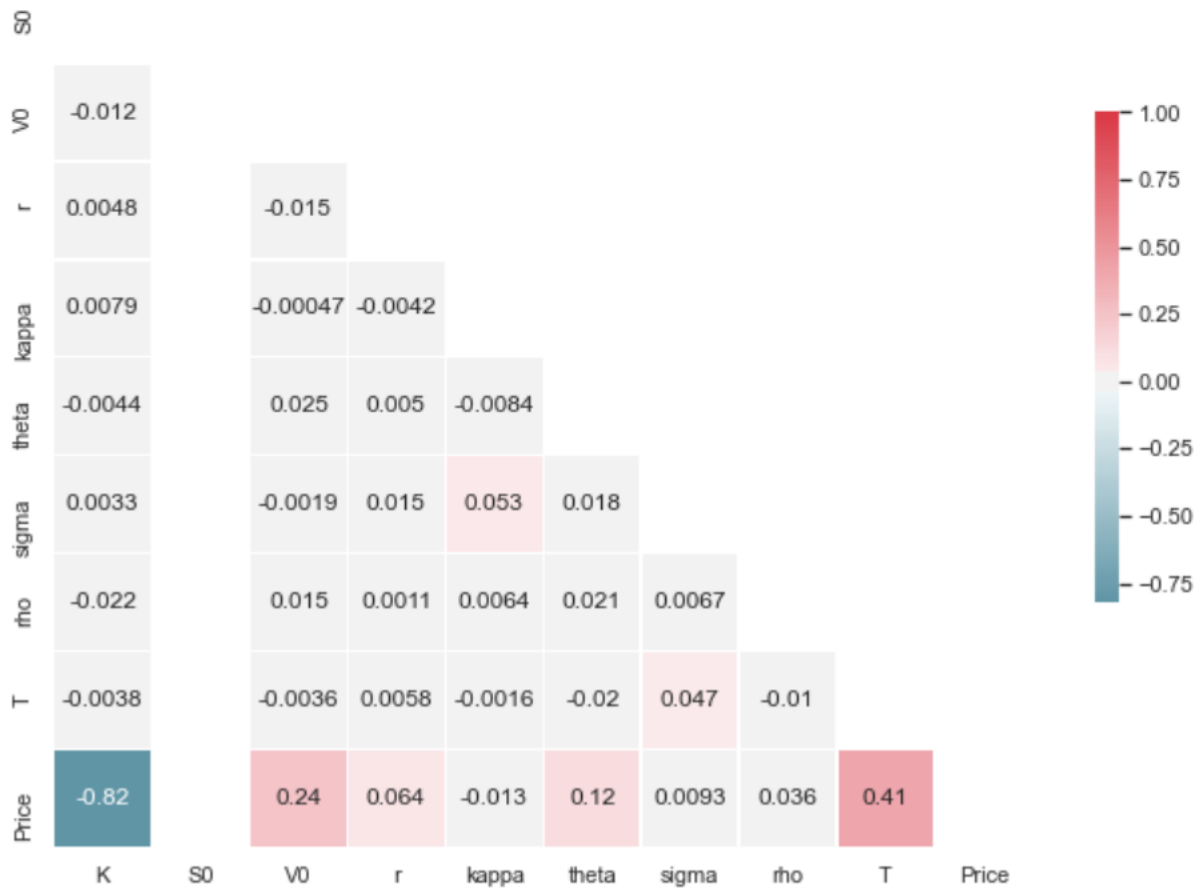

FIGURE 4 – Correlation plot of our dataset

5.2 Modelization

On this section we are going to compute various machine learning algorithm in order to find the best prediction for our target variable.

5.2.1 Baseline model : Ordinary Least square method

Our baseline model (Price accroding to all the explanatory variables) gives us an adjusted R-squared of 0.910.Then approximately 90% of the observed variation can be explained by the model's inputs. We plot our estimated target with respect to our real target and then superimpose the first bissextrix (y=x). If the values are perfectly predicted, we expect to see points along the y = x line.The plot(see

section plot below) shows us that our estimation is quite different from the real values. We use the statsmodels.api package to build the Ols regression model.

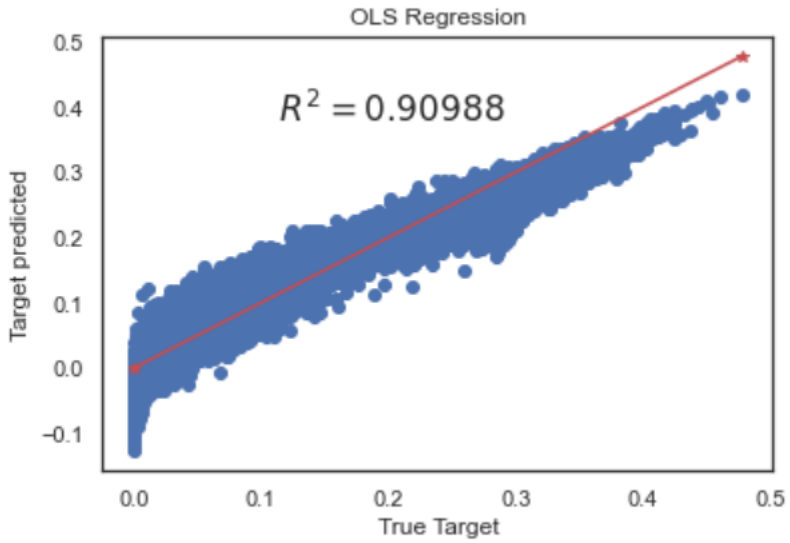

FIGURE 5 – Ols Regression

So we are going to try another algorithms to see if we can obtain a better fit.
In the following, we split our data into two subsets, one train set (80% of the training dataset) on which to develop our models and a testingset (20%) on which we will evaluate our models.

5.2.2 Ridge

We use the Ridge() function available in the sklearn.linear_model.Ridge package to find the best λ. This function does k-fold cross-validation, produces a plot, and returns a value for the best lambda value and the coefficient of determination R^2 of the prediction which are equal in our case to λ_{ridge} = 0.01 and R^2_{ridge} = 0.91 respectively.

```
{'alpha': 0.01}
R2 score 0.9117699761089298

Out[32]: Text(0.5, 1.0, 'Ridge Regression')
```

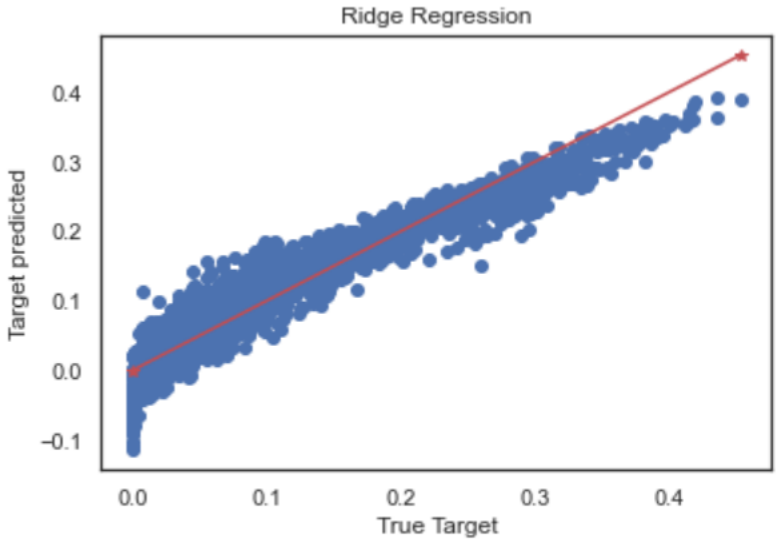

FIGURE 6 – Ridge Regression

5.2.3 Lasso

We use the Lasso() function available in the sklearn.linear_model.Lasso package to find the best λ. This function does k-fold cross-validation, produces a plot, and returns a value for the best lambda value and the coefficient of determination R^2 of the prediction which are equal in our case to $\lambda_{lasso} = 1e^{-08}$ and $R^2_{lasso} = 0.91$ respectively.

```
{'alpha': 1e-08}
R2 score 0.9117709072224445

Out[33]: Text(0.5, 1.0, 'Lasso Regression')
```

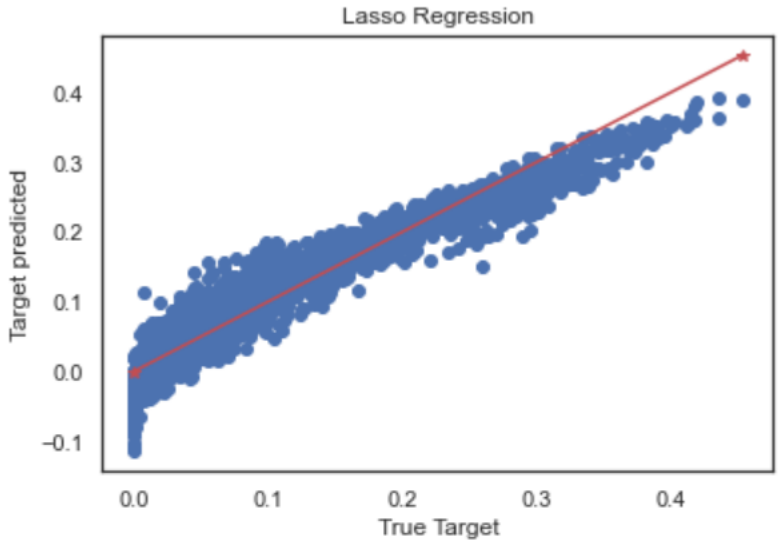

FIGURE 7 – Lasso Regression

5.3 k-nearest neighbor(KNN)

A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the same distance functions as KNN classification.We tried to find the most optimal k value by choosing different ones randomly and repeating the test multiple times till we find the best R^2 score and a minimal error.

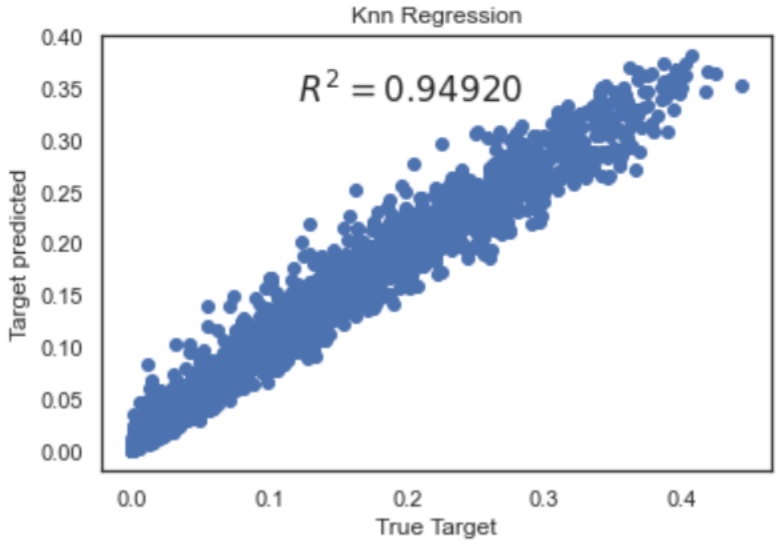

FIGURE 8 – Knn Regression

5.4 Multivariate Polynomial regression

Polynomial Regression is a form of linear regression in which the relationship between the independent variables and the target is modeled as an nth degree polynomial. Polynomial regression fits a nonlinear relationship between the value of features and the corresponding conditional mean of the dependent variable.We tried to find the most optimal degree value by choosing different ones randomly and repeating the test multiple times till we find the best R^2 score and a minimal error.
If we apply polynomial transformation of degree 2, and compute the MSE and R^2 score of the quadratic plot gives :

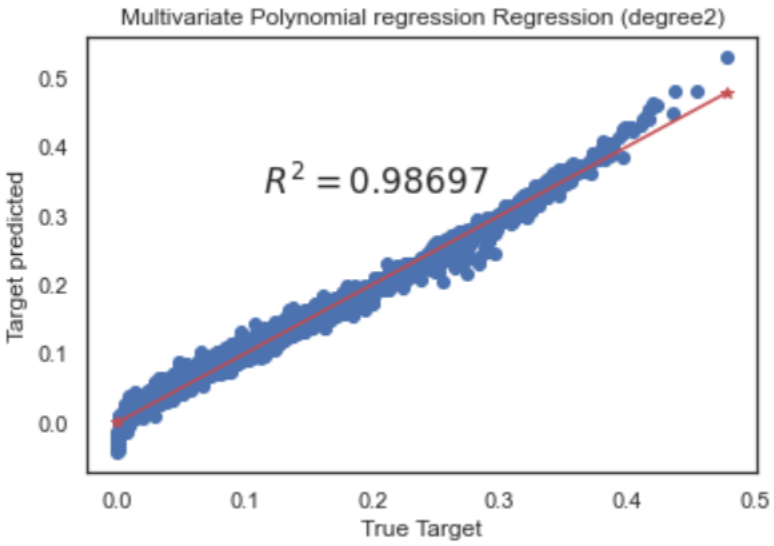

FIGURE 9 – Multivariate Polynomial Regression (degree 2)

If we try to fit a quartic curve (degree=4) to the dataset, we can see that it passes through more data points than the quadratic and the linear plots.We can see that RMSE has decreased and R^2-score has increased as compared to the linear line.

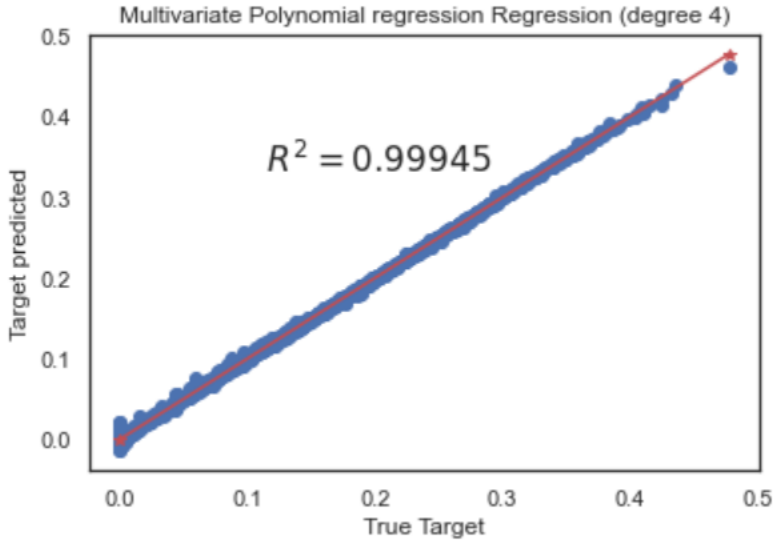

FIGURE 10 – Multivariate Polynomial Regression (degree 4)

Note that the best predictions was obtain with the Multivariate Polynomial regression at the moment.

5.5 Artificial Neural Networks

5.5.1 Methodology

In this section, we present a neural network to approximate a function for financial models. The procedure comprises two main components, the generator to create the financial data for training the model and the predictor (the ANN) to approximate the option prices based on the trained model. The data-driven framework consists of the following steps :

Algorithm 1 Model framework

— Generate the sample data points for input parameters,

— Calculate the corresponding output (option price or implied volatility) to form a complete data set with inputs and outputs,

— Split the above data set into a training and a test part,

— Train the ANN on the training data set,

— Evaluate the ANN on the test data set,

— Replace the original solver by the trained ANN in applications.

5.5.2 Hyper-parameters optimization

Sometimes it can be difficult to choose a correct architecture for Neural Networks. Usually, this process requires a lot of experience because networks include many parameters. Let's check some of the most important parameters that we can optimize for the neural network :

— Number of layers

— Different parameters for each layer (number of hidden units, filter size for convolutional layer and so on),

— Type of activation functions

— Parameter initialization method

— Learning rate

— Loss function.

We opt for the Random Search to select the hyperparameter.The simplest algorithms that for hyperparameter optimization is a Grid Search. The idea is simple and straightforward. We just need to define a set of parameter values, train model for all possible parameter combinations and select the best one. This method is a good choice only when model can train quickly, which is not the case for typical neural networks.The idea of Random Search is similar to Grid Search, but instead of trying all possible combinations we will just use randomly selected subset of the parameters. Instead of trying to check 100,000 samples we can check only 1,000 of parameters. It takes too long to run hyperparameter optimization.

There are two stages to complete the hyper-parameter optimization. During the model selection process, over-fitting can be reduced by adopting the k-fold cross validation as follows.

Algorithm 2 k-fold cross validation

- Split the training data set into k different subsets,
- Select one set as the validation data set,
- Train the model on the remaining k-1 subsets,
- Calculate the metric by evaluating the trained model on the validation part,
- Continue the above steps by exploring all subsets,
- Calculate the final metric which is averaged over k cases,
- Explore the next set of hyper-parameters,
- Rank the candidates according to their averaged metric.

TABLE 1 – The setting of random search for hyper-parameters optimization

Parameters	Options or Range
Activation	ReLu, tanh, sigmoid, elu
Dropout rate	[0.0, 0.2]
Neurons	[200, 600]
Initialization	uniform, glorot_uniform, he_uniform
Batch normalization	yes, no
Optimizer	SGD, RMSprop, Adam
Batch size	[256, 3000]

TABLE 2 – The selected model after the random search

Parameters	Options
Hidden layers	4
Activation	eLu
Dropout rate	0.0
Neurons	400
Initialization	glorot_uniform
Optimizer	Adam
Batch size	3000
Batch size	1024

5.5.3 Numerical Results

We show the performance of the ANNs for solving the financial models, based on the following accuracy metrics (which forms the basis for the training),

$$\text{MSE} = \frac{1}{n}\sum (y_i - \hat{y}_i)^2 \tag{34}$$

where y_i is the actual value and is the ANN predicted value. The MSE is used as the training metric to update the weights, and all above metrics are employed to evaluate the selected ANN. For completeness, however, we also report the other well-known metrics,

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{35}$$

$$\text{MAE} = \frac{1}{n}\sum |y_i - \hat{y}_i| \tag{36}$$

$$\tag{37}$$

We list below the performance of the ANN on the wide test data set.

TABLE 3 – Heston-ANN performance on the test data set

Heston ANN	MSE	RMSE	MAE	R^2
Training	$4.9e^{-05}$	$6{,}99e^{-03}$	$5{,}25e^{-03}$	0.99995
Test	$5.99e^{-05}$	$7{,}73e^{-03}$	$5{,}66e^{-03}$	0.99994

In order to build more robust models, it is common to do a **k-fold cross validation** where all the entries in the original training dataset are used for both training as well as validation. Also, each entry is used for validation just once. We built a 5-fold cross validation model and return the results in each step as shown below .The final round metric has reduced as compared to last time.

```
Fold #1
Fold score (RMSE): 0.0008007587939211456
Fold  R2: 0.9999399194545004
Fold #2
Fold score (RMSE): 0.0008172394858433223
Fold  R2: 0.9999398467573826
Fold #3
Fold score (RMSE): 0.0011448931632047194
Fold  R2: 0.9998834276585244
Fold #4
Fold score (RMSE): 0.0008893074531543141
Fold  R2: 0.999925968876442
Fold #5
Fold score (RMSE): 0.0009420248058463962
Fold  R2: 0.9999202085686865
Final, out of sample score (RMSE): 0.0009271396328714913
Fitting time:  543.2054440975189
```

FIGURE 11 – Cross validation ANN

We can see on the following figure that the scatter plot of the predicted values with respect to the real values is alongside the first bisector.

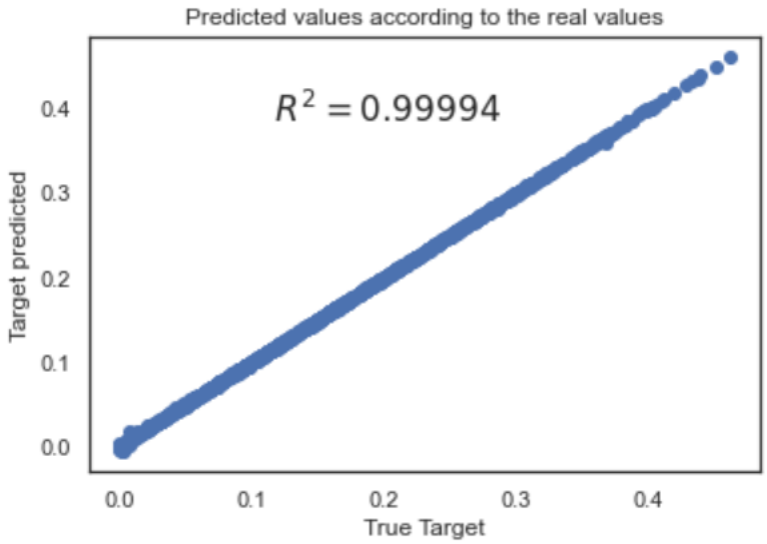

FIGURE 12 – Cross validation ANN

5.5.4 The influence of some hyperparameter of the model

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches.

We create line plots that show epochs along the x-axis as time and the error(Rmse figure (), log(MSE) figure())of the model on the y-axis. These plots are called learning curves.

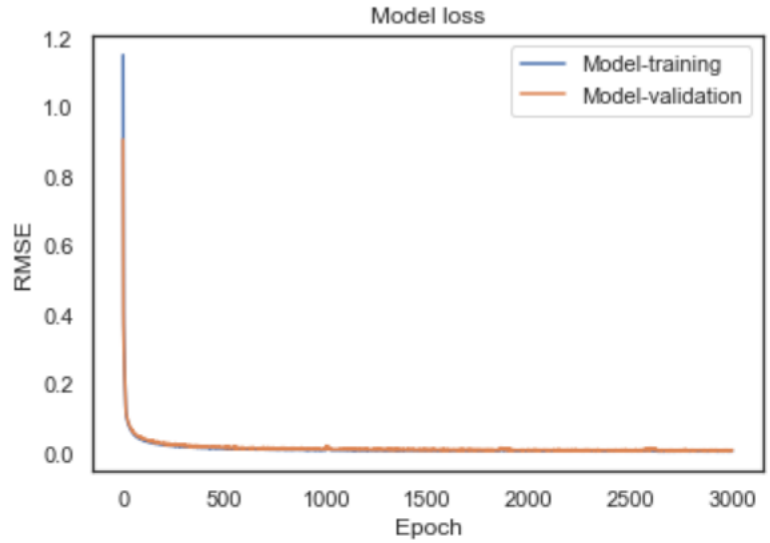

FIGURE 13 – Loss(RMSE) of the neural networks according to the value of the epoch parameter

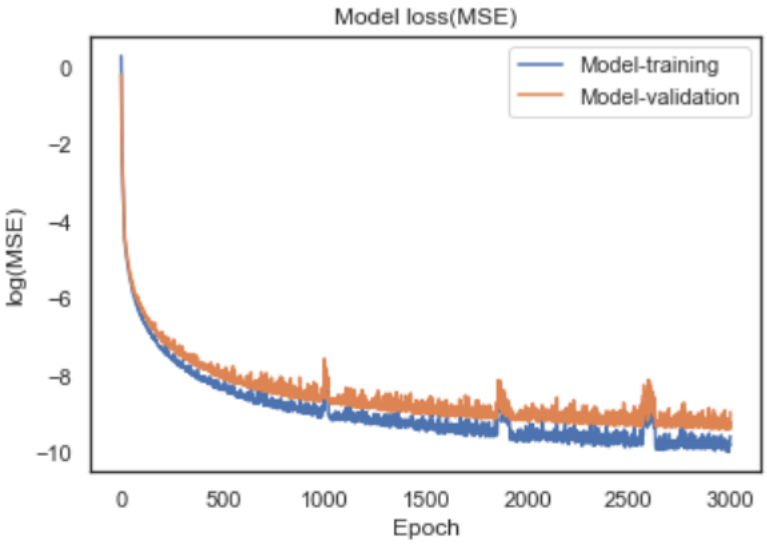

FIGURE 14 – Log(MSE) of the neural networks according to the value of the epoch parameter

These plots show that the model is suitably fit to the training dataset.

5.6 XGboost

Before building the model, we should choose the tuning parameters that XGBoost provides. The most common ones are :

— n_estimators : number of trees to build

— objective : determines the loss function to be used like reg :squarederror for regression problems,

— max_depth : determines how deeply each tree is allowed to grow during any boosting round.

— colsample_bytree : percentage of features used per tree. Low value can lead to overfitting.

Also, to maintain reproducibility of the results, a random_state is also assigned.We tried to find the most optimal Hyperparameter by choosing different ones randomly and repeating the test multiple times till we find the best R^2 score and a minimal error.

```
-----------------------------------------------------------------------
Hyperparameters :{objective =reg:squarederror, colsample_bytree =1,
                  learning_rate = 0.1,n_estimators = 10000, alpha=0}
-----------------------------------------------------------------------
test r2 score:  0.9960886394288339
-----------------------------------------------------------------------
mean_squared_error: 0.00013590877001805161
```

Out[97]: `Text(0.5, 1.0, 'XGboost Regression ')`

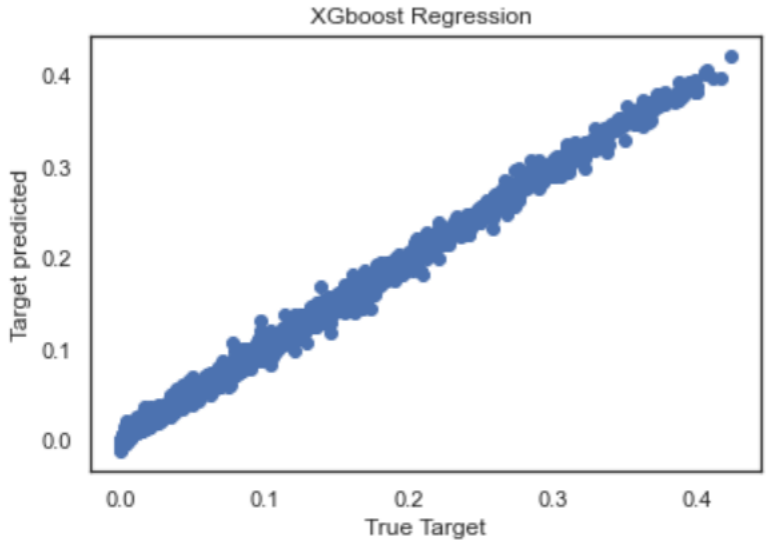

FIGURE 15 – XGboost Regression

In order to build more robust models, it is common to do a **k-fold cross validation** where all the entries in the original training dataset are used for both training as well as validation. Also, each entry is used for validation just once. XGBoost supports k-fold cross validation via the cv() method. We built a 10-fold cross validation model by invoking XGBoost's cv() method and store the results in a cv_results DataFrame. cv_results contains train and test MSE metrics for each boosting round.The final boosting round metric has reduced as compared to last time and came out to be around 0.000091.

```
------------------------------------------------------------------------
   train-rmse-mean  train-rmse-std  test-rmse-mean  test-rmse-std
0         0.337957        0.000294        0.337964       0.002678
1         0.304532        0.000264        0.304575       0.002418
2         0.274439        0.000240        0.274467       0.002207
3         0.247346        0.000214        0.247377       0.001993
4         0.222953        0.000193        0.223008       0.001834
------------------------------------------------------------------------
49    0.000091
Name: test-rmse-mean, dtype: float64
```

FIGURE 16 – K-fold Cross Validation using XGBoost

5.6.1 Comparaision of the Regression models

To compare these different methods we will use the k-fold method,(Algorithm 2) by dividing our initial training database into two subsets, one train set on which to develop our models and a testing set (on which we will compare which models give the best results and then apply that model on the original test dataset from kaggle. Cross-validation is the process of assessing how the results of a statistical analysis will generalize to an independent data set. If the model has been estimated over some, but not all, of the available data, then the model using the estimated parameters can be used to predict the held-back data. If, for example, the out-of-sample mean squared error, also known as the mean squared prediction error, is substantially higher than the in-sample mean square error, this is a sign of deficiency in the model. For each model,we split randomly the initial dataset in k folds. The k-1 folds are used to train our model (train dataset) and the remaining observations are used as the 'Test' data set). We use the training data set to estimate the parameters of the model and compute the RMSE of the model. Given the previous model,we use the test data set to compute the RMSE to evaluate the

performances of the model. We compare the results obtained for each model with help of boxplots. Here the best model is the model with the smallest error : ANN .

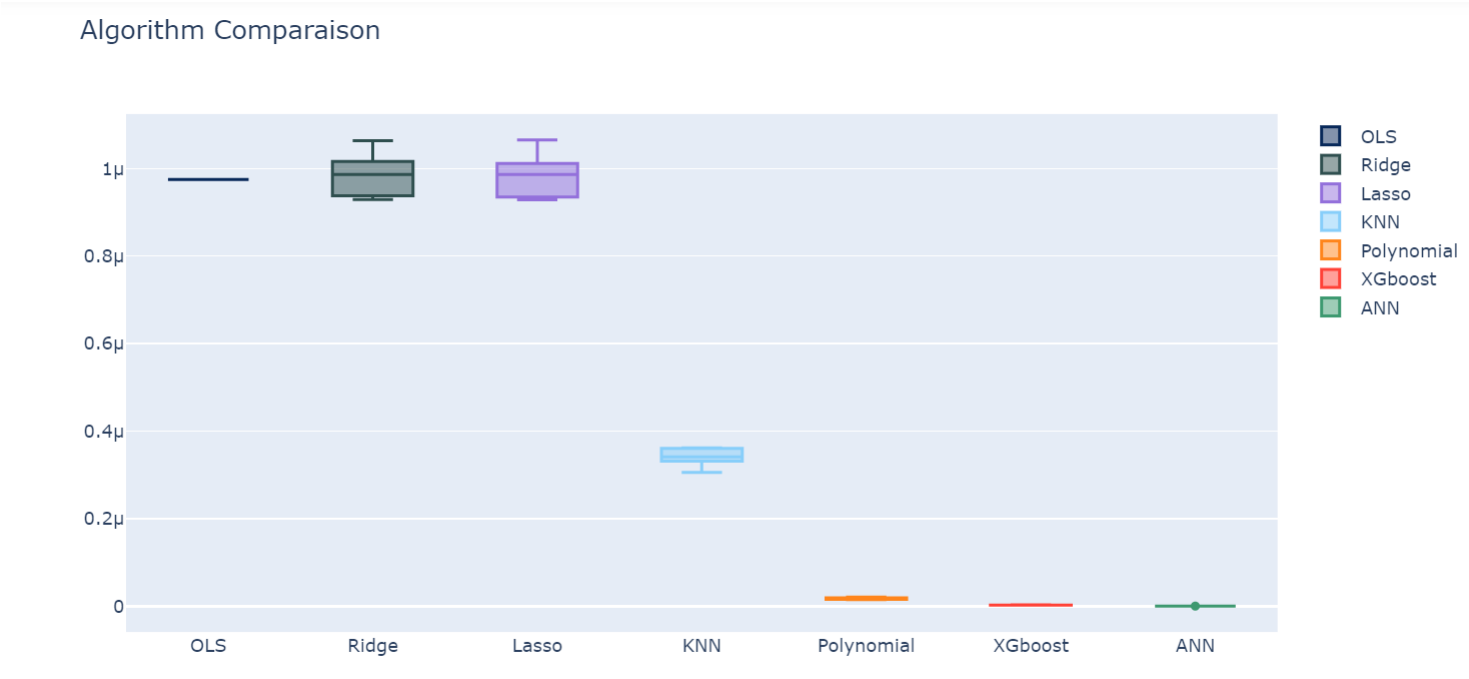

FIGURE 17 – Model comparaison

6 Statistical methods for implied volatility calculation

We were challenged to build a model that approximate the Black-Scholes implied volatility. First we implemented the pricer from the black-scholes model using the closed formula we have and used that function to create the dataset. So we chose a range of value for each parameter (σ, r, S_0, K, T) then we computed the price using the blacksholes formula. We obtined a data set composed of 6 variables. We then took σ which is the volatility of the model as our target variable and the price($V_m kt$) and the other parameters are considered as the explanatory variables.The data set composed of 10 000 observations of 10 initial variables with the target variable being the **volatility**.

	V_market	S0	r	T	K	Volatilite
0	2.241121e-02	1	0.083085	0.855814	2.174112	0.526896
1	7.302293e-02	1	0.094361	0.922862	1.583641	0.504969
2	9.342592e-02	1	0.028980	0.660503	1.555847	0.707901
3	1.548390e-01	1	0.072944	1.079435	2.480664	0.948515
4	7.441252e-03	1	0.044679	0.225593	2.050979	0.819770
...	...	...	...	...	...	...
9995	3.466363e-10	1	0.029368	0.627661	1.482769	0.086849
9996	3.579835e-05	1	0.095176	0.678029	2.449530	0.299390
9997	9.416079e-02	1	0.031120	0.983248	1.530036	0.563648
9998	1.339155e-01	1	0.095320	0.715998	2.008919	0.962059
9999	8.117761e-02	1	0.033024	0.596123	2.292737	0.989975

10000 rows × 6 columns

FIGURE 18 – Data set

First, we started by cleaning up our data set. That means getting rid of missing values and any observation that seem like an outlier. Secondly, we will look at the different relation between the feature variables and the target variable. Thirdly we are going to split our data set intot a trainign cset composed of 80% of the observations and a testing set compose of 20% of the dataset we then compute various machine learning algorithms on the training set going from a simple linear model to more sophisticated algorithms such as neural network algorithms and validate those models using the testing set . Finally, we will compute the boosting algorithm and see if it improve our predictions.

6.1 Looking into the relation between the variables

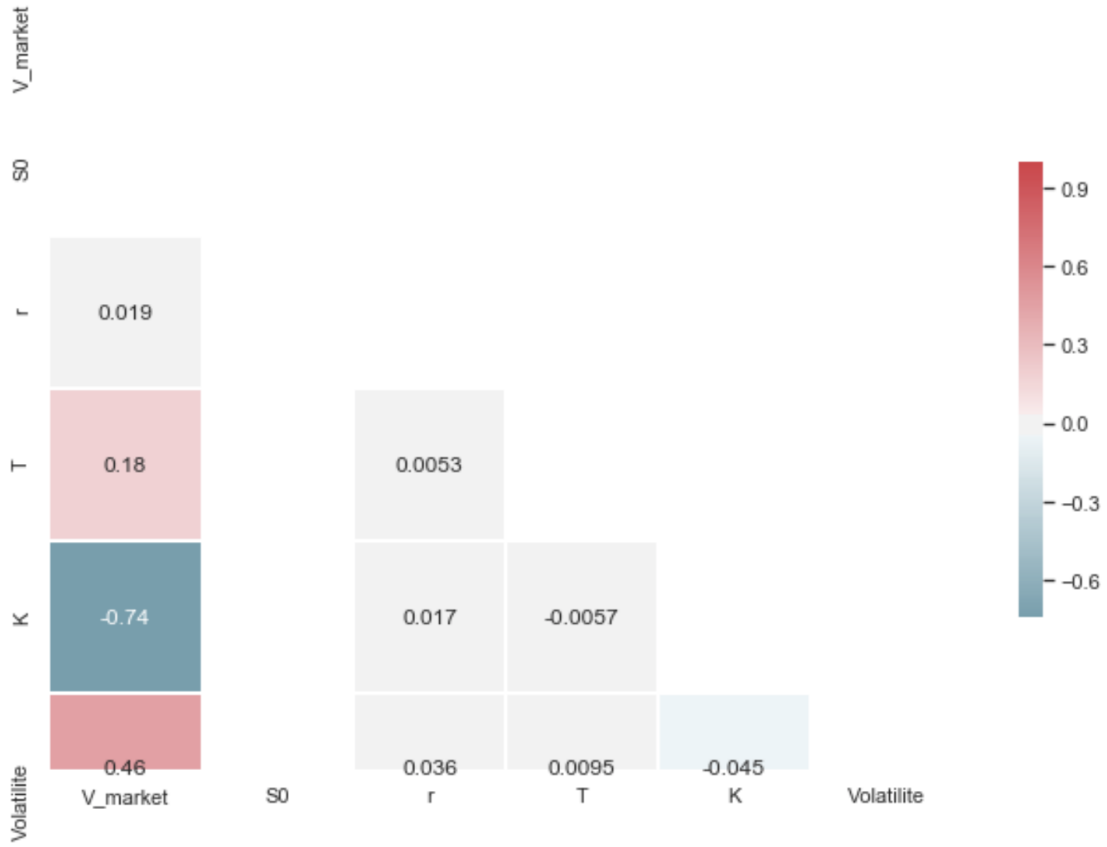

FIGURE 19 – Correlation plot of our dataset

We see in this correlation plot shows that the target variable is mainly linked to the pricce $V^m kt$ which is understandable.

6.2 The results

In this section we are going to present the result we obtained from each of the previous algorithm

6.2.1 Ordinary Least square method(OLS)

When we compute the OLS regression on our dataset we obtained an R-squared of 0.398 which means that our model doesn't really fit the true variable

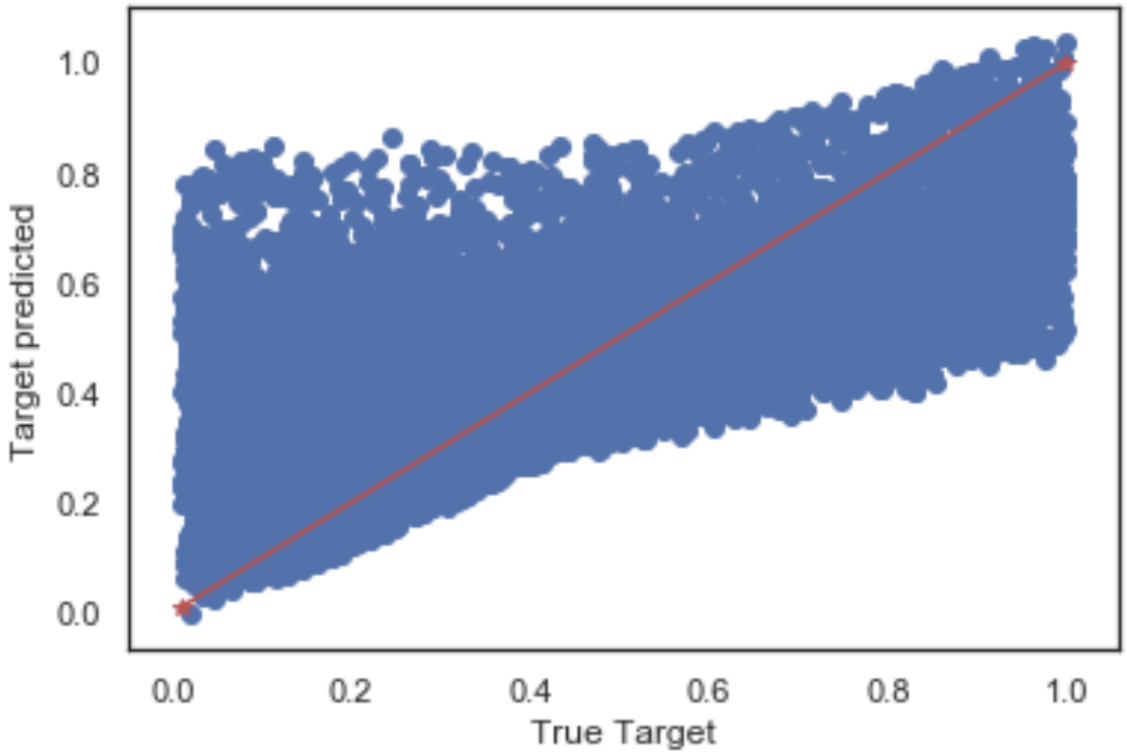

FIGURE 20 – Result of OLS Regression

On the figure abive we plotted the true target according to the estimated target anw superimposed the first bissector (y=x). If the estimation was good we expect to see the point alogi that first bissectrix but as you can see it the point are scattered around the line **Conclusion : The ols regression does not fit our target variabl which is uderstandable since we know the relatio betweeen the target and the parameters is far from linear.**

6.2.2 Penalized regression : Ridge and lasso

We also tried penalized regression methods but they weren't more succesful than the basic OLS . We obtained an R^2 of 0.388 for Ridge and -0.049 for Lasso

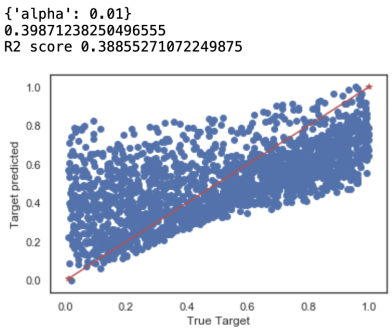

FIGURE 21 – Result of ridge regression

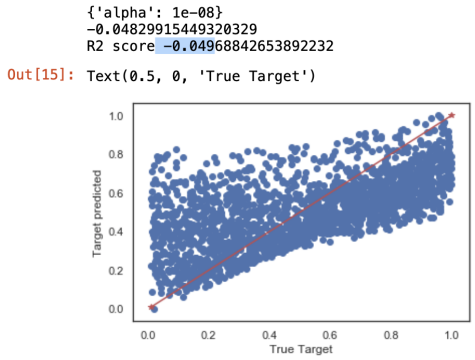

FIGURE 22 – result of lasso regression

6.2.3 k-nearest neighbor(KNN)

A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the same distance functions as KNN classification.We tried to find the most optimal k value by choosing different ones randomly and repeating the test multiple times till we find the best R^2 score and a minimal error. Even tho the KNN algorith is kind of popular on estimating linear and non linear functions it didn't work at all on our data set

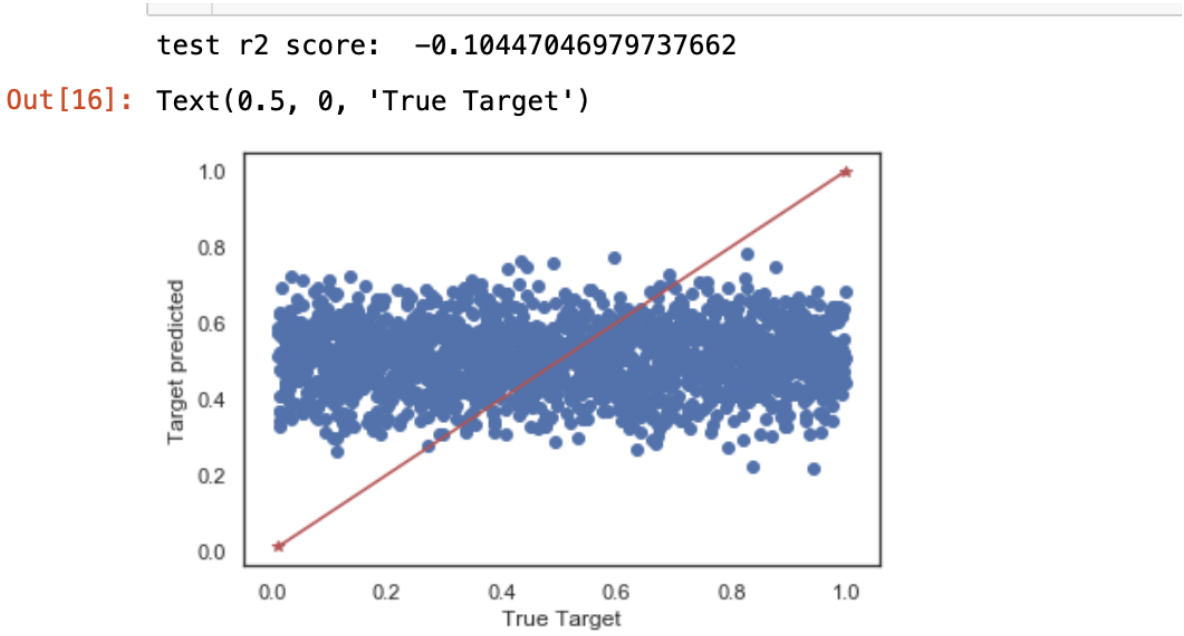

FIGURE 23 – Result of KNN

As you can see on the figure the points are not even close to the first bisector and the R^2 of the model is -0.1 which means the model doesn't fit at all .

6.2.4 Multivariate Polynomial regression

Polynomial Regression is a form of linear regression in which the relationship between the independent variables and the target is modeled as an nth degree polynomial. Polynomial regression fits a nonlinear relationship between the value of features and the corresponding conditional mean of the dependent variable.We tried to find the most optimal degree value by choosing different ones randomly and repeating the test multiple times till we find the best R^2 score and a minimal error. We found out that the most accurate estimation was obtained with a polynomial regression of degree 4

```
test r2 score:  0.8808090763590698
mean_squared_error: tf.Tensor(0.009462763505842678, shape=(), dtype=float64)
```

```
Out[29]: Text(0.5, 1.0, 'Multivariate Polynomial regression Regression (degree 4)')
```

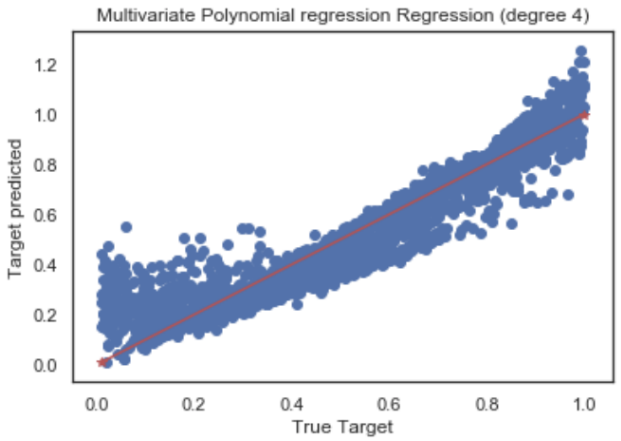

FIGURE 24 – Multivariate Polynomial Regression (degree 4)

Note that the best predictions was obtain with the Multivariate Polynomial regression at the moment. The polynomial regression gives us a model that fits better our dataset compares to the 3 previous one. Now we're gonna move on to the use of neural networks for the function approximation

6.2.5 Artificial neural networks for implied volatility computation

We followed the same methodology that was explained previously for the option pricing. We trained a four layer neural network using the keras library of python and did a cross validation to check the stability of our model Here are the results we obtained.

```
Fold #1
Fold score (RMSE): 0.06388957739951688
Fold  R2: 0.9480116600717166
Fold #2
Fold score (RMSE): 0.05997588416695303
Fold  R2: 0.9556377442336277
Fold #3
Fold score (RMSE): 0.049856826365640726
Fold  R2: 0.9697782801098493
Fold #4
Fold score (RMSE): 0.055668465561780435
Fold  R2: 0.9625180222793849
Fold #5
Fold score (RMSE): 0.053303177938707244
Fold  R2: 0.9637946692277402
Final, out of sample score (RMSE): 0.056753669050588956
```

FIGURE 25 – The different result obtained by cross validation for the RMSE end the R^2 of the ANN estimator

The K-fold enables us to conclude that our neural network has a RMSE of 0.05 and a R^2 of 0.94 which mean that the neural has indeed done a pretty accurate approximation linking the target variable(implied volatility) to the explanatory variables We can see on the following figure that the scatter plot of the predicted values of implied volatility with respect to the real values is alongside the first bisector.

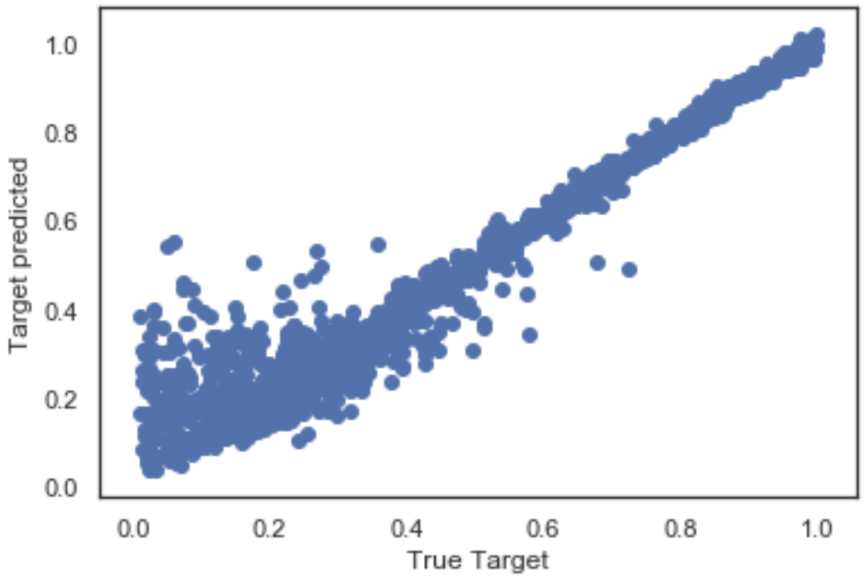

FIGURE 26 – plot of predicted values of the implied volatility according to the real values

6.2.6 The influence of some hyperparameter of the model

In Deep Learning, an epoch is a hyperparameter which is defined before training a model. One epoch is when an entire dataset is passed both forward and backward through the neural network only once. It can take thousands of epochs for your backpropagation algorithm to converge on a combination of weights with an acceptable level of accuracy. Remember gradient descent only changes the weights by a small amount in the direction of improvement, so backpropagation can't get there by running through the training examples just once. So we noticed that the loss on bothe the training and testing dataset is decrease as the value of the epch parameter increase as it is shown in the following figure

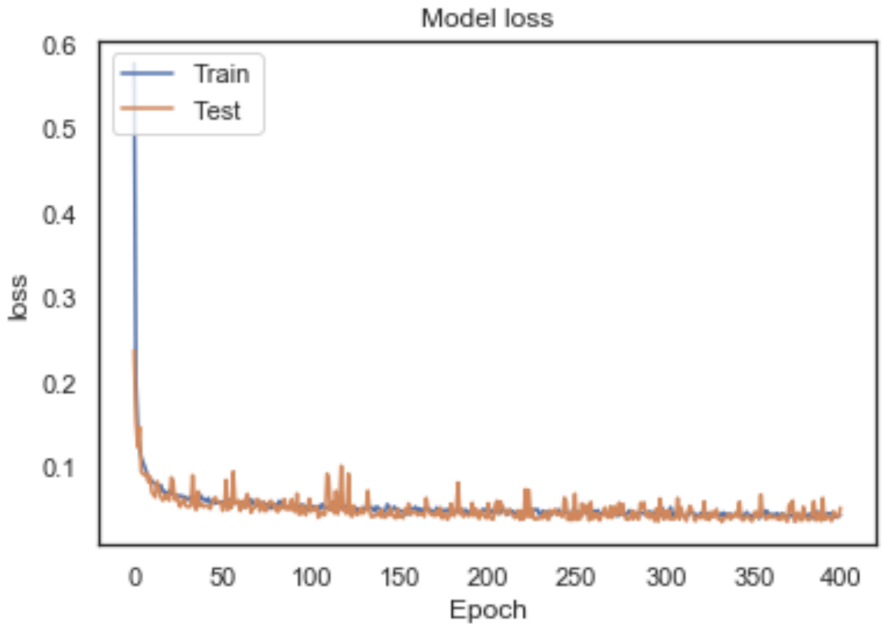

FIGURE 27 – Loss of the neural networks according to the value of the epoch parameter

7 conclusion

To conclude, the result we obtained showed that effectively we can use neural networks for pricing and computing the implied volatility of financial models. The approximation we obtained are really close to the real values. Even tho the model takes a lot of time train especially for the creation of the database, once the model is trained it can be efficiently used on a new dataset reducing considerably the calibration time.

8 Bibliography

[1] B. Horvath, A. Muguruza, and M. Tomas, "Deep learning volatility," *Available at SSRN 3322085*, 2019.

[2] S. Liu, C. W. Oosterlee, and S. M. Bohte, "Pricing options and computing implied volatilities using neural networks," *Risks*, vol. 7, no. 1, p. 16, 2019.

[3] L. B. Andersen, "Efficient simulation of the heston stochastic volatility model," *Available at SSRN 946405*, 2007.

[4] N. Moodley, "The heston model : A practical approach with matlab code, bachelor thesis," *University of the Witwatersrand, Johannesburg, South Africa, online at http ://www. math. nyu. edu/˜ atm262/fall06/compmethods/a1/nimalinmoodl ey. pdf, last access*, vol. 1, 2009.

[5] D. Lamberton and B. Lapeyre, *Introduction to stochastic calculus applied to finance*. CRC press, 2007.