COMP6212 Assignment 3 : Shakib-Bin Hamid 25250094 sh3g12

Black-Scholes arrived at a closed form solution of the derivative pricing problem under some strict conditions. Some of these conditions can be eased by creating a binomial lattice based on probabilities on returns. However, these are still parametric methods and sensitive to the underlying stochastic process for $S(t)$, the stock price over time. Misspecification of it will lead to systematic errors in the option's price calculation, i.e. the performance of the parametric models is closely tied with the ability to capture the dynamics of underlying asset or stock.

On the other hand, learning networks like RBF (Radial Basis Function), MLP (Multi Layer Perceptron) etc. 'learn' the underlying dynamics based on training data and target outputs. They can adapt to the structural changes to the data generating process. It is true that such methods are not needed if the option in question is very well understood, or a new option is made and cannot be captured as a combination of other options or if there is not enough training data. However, these are rare circumstances indeed.

In this exercise we are primarily concerned with RBF networks. It has been shown that RBFs have the best approximation property, i.e. there is always a choice for the parameters that is beter than any other possible choice - not shared by MLPs. The paper poses the following challange - " if option prices are truly determined by the Black-Scholes formula exactly, can learning networks like RBF 'learn' the formula?". In other words, if we generated a dataset of options prices given strike prices, time to maturity and stock price, and trained a RBF on it, then will it generate the same prices on unseen data as Black-Scholes' formula would? If it does, then we can use this non-parametric model in future, even without the strict assumptions in Black-Scholes.

I began by creating the training dataset from Black-Scholes formula. For $T/4+1$ to $3T/4$ days, I calculated the historical volatility as the standard deviation of log returns from the previous $T/4$ days for each of the call options. I fixed interest rate, $r = 0.06$. Given the strike price and FTSE 100 index as the underlying asset price, I used Matlab's Financial Toolbox's `blsprice` to calculate the Black-Scholes call option price and `blsdelta` to calculate the deltas for those days. I then normalised the index, S and option price, C by the strike price X. Time to maturity, $T-t$ was also calculated for the days. This is my training dataset. The final $T/4$ days' data was prepared similarly as the validation set to test how closely the RBF network would perform on unseen data.

The model for the RDF network is $c = \Sigma_{j=1}^{J}\lambda_j\phi_j(\boldsymbol{x}) + \boldsymbol{w}^t\boldsymbol{x} + w_0$, where the nonlinarity is captured in the basis function, $\phi(\boldsymbol{x}) = \sqrt{(\boldsymbol{x}-\boldsymbol{m}_j)\Sigma(\boldsymbol{x}-\boldsymbol{m}_j)^t + b_j}$, i.e. the Mahalonobis distance if the bias term is set to 0. We will use $J = 4$ or in other words, the RBF network will have 4 RBF compute nodes which will take inputs and feed into the model's equation.

I began with preparing the design matrix $\boldsymbol{\Phi}$. The input data was $\boldsymbol{x} = [S/X(T-t)]^t$, i.e. the normalised strike price and the time to maturity for the training days. The 4 centres for Mahalonobis distance were found using `fitgmdist` which fit a Gaussian Mixture Model with 4 sub gaussian models. Note that on some runs of the system, the function failed to converge within 100 iterations. Then the first 4 columns of $\boldsymbol{\Phi}$ were calculated as the Mahalonobis distance between the input vector and the centres. After that the input data was inserted unchanged as the next 2 columns (to calculate the linear term in the RBF model). Finally a column of ones was inserted in the design matrix at the last column (for the constant term in the RBF model).

Once the design matrix was ready for both the training and validation set of our experiment, the weights of the model, a 7 item column vector $\boldsymbol{\lambda}$ was calculated using `pinv` and *only* the training data. `pinv` (Moore-Penrose Pseudo-Inverse) with 0.1 tolerance was used so as to not fall into the trap of ill-conditioned matrix $\boldsymbol{\Phi}$. Finally, this $\boldsymbol{\lambda}$ was used on the validation set's design matrix $\boldsymbol{\Phi}_{val}$ to calculate C/X for the validation set. I followed the same steps to find the deltas for the validation set by using the weights found from a 4 node RDF network trained on the training deltas.

References

[1] J. Hutchinson, A. Lo, and T. Poggio, *A nonparametric approach to pricing and hedging derivative securities via learning networks.* The Journal of Finance, vol. 49, no. 3, pp. 851889, 1994.