Semaan Youri Serro Julien

VBA projet: The perceptron in finance

In this essay, we are going to explain several choices we made in the way we coded our how our code is indented in addition to all the comments already on our code.
The program can be seen in two different sheets, the AR-GARCH Model and the perceptrons/ multi-layer perceptrons.
The dynamic Implementation of the perceptron xlsm is the same one as the normal however you can choose several useful values inside.

• Perceptron

We first here calculated the last row used on the graph to have something dynamic, and we redimensionnes the closing prices array according to that by removing 1 since there is a header in the excel sheet.
We then declared all the variable useful for our code and we putted rolling window equal to 20 which is something we preferred to change only in the code and not in an interface since all our model is based on a rolling window of 20.

• Functions

We then defined several functions such as returns() which computes the daily stock returns by taking in argument an array returnsofprices() and fills it according to the closingprices().

There is also the forwardreturn() which is basically the same than the return but we cheat a little bit we 'predicting' the forward return value and then change to 1 or 0 according if it is positive or negative. The binary value is crucial for our perceptron model.

Then the simplemovingaverage(returnofprices(), sma, sma20(), rollingwindow). First of all we redimension the sma20() array according to the length of the returnofprices() array, we remove the rolling window since we can't compute the sma20 during the last 20 values. We just compute all the sma20 and then use them to have the general SMA.

Finally there is the bollingerbands(returnofprices(), sma, sma20(), bb(), rollingwindow). Which takes all of those arguments and changes the bb() array which corresponds to the lower and upper band of the bollinger band.

• Simple Perceptron

In this class we have several properties such as mLearningrate, mNbofiterations, mLoss, mBias, mWeight().

- Functions

Our First function is the Sub Initialization() which assigns random weights between O to 1 to our array defined in our property.

Then we have the ForwardComputation(inputs(), method) That is the first part of the 'loop' we just simply do the matricidal multiplication of the weights and the inputs and apply it to any of the activation method chosen (here method is an integer that corresponds respectively to the number of the method presented in the subject)

Finally we have our BackwardComputation(inputs(), input_values(), real_output(), method) the inputs() and method argument are used for the ForwardComputation which is called inside the Backward Computation. We first compute our training_output value in the function the use it for the loss and the bias. And then adjust the weights.

- Main

We first declare a lot of variables that we are going to use for the next steps of our program, then we declare our perceptron and its defat parameters. Here our training data is on 80% of the data, however we can change this by changing the length. A condition changes the length to 0.8 of it is under 0.1 or above 0.9, it does not make any sense if not. We didn't choose 0 and 1 if not it is simply just using the full data.

We then redimension the training and the test and fill it according to the parameter length that was used. We also redimension all the other variables due to the number of iterations.

Then we initialise the weights and call the functions defined before for training and test, after this we fill up the training input and test which are arrays (lower band, sma, upper band) .

Finally, We call the BackwardComputation function as many times as there is iterations and we stock it at each time, we did it here for the training and for the sigmoid activation. However to do it for another activation function we just have to change the method number.

- Comments

We are sorry to say we did not have enough time to do the confusion matrix, the weight values corresponds to the importance of each input.

We plotted the graph for the 3 different activation function to do this we just have to change the method number. In 3 of them, the perceptron converges, thus It converges increasingly respectively to the 3 methods used. And we didn't have enough time to do the 2 different initialisation methods which surely helps to converge better and quicker

• AR-GARCH

All the program here is in the main since everything is directly on excel.
We first declare all the variables used further in our program. set our workbook and call randomise. In our program we changed the last row to 100 because it takes too long, to have all the row used, you just have to comment lastrow=100 and uncomment the code before.

We then assign randomly all our values (sigma, alpha, beta, eta) since they are going to change later one with the solver

For sigma, we simply use the formula given and we make sure to use the function .Formula so that the values in the cells depend of each other so that the solver will be able to make them change, because if we just put values in the cells and not formulas, the solver won't be able to change constant values.

We use the formula as well for the predicted price return.

For the residual of the Ar part we we simply substrat the price return and the predicted price return.

For epsilon it is residual/sigma.

And finally for the log-likehood we use the formula as well.

When all this is done we have our excel that is filled, and we just have to use the solver: click on the cell N3(log-likehood), write that we want the min by making the changes on all the values of the K column and adding a constraint which is that our cell N4(variance of epsilon) equals 1. Everything is moved when we choose the n, the formulas are adapted and the display on the sheet changes as well.

• MLP

• Functions

We basically have the same properties than the simple perceptron except that the weights are two dimensions.

Everything else is basically the same except that inputs and weights are in two dimension.

• Main

We declare the multi-layer perceptron and the variables useful for the rest of the program. We then initialise the weights, and n correspond to the number of inputs we have at the beginning.

We then redimension our variables and fill up our inputs for the MLP,

Finally we just call the backward loop like for the simple perceptron but we didn't have the time to update the values correctly like in the AR-GARCH method.