Machine Learning In Depth Analysis

The machine learning has been the more intriguing portion of this project. The goal of prediction in the stock market went from something large to something quite small, that is, from predicting industry price movement based on another industry to just simply predicting a single company's prices over a small period of time. As always, the goal of prediction is to compare errors and decide on the best model. Here the definitive metric was Mean Absolute Percent Error.

While we were exploring the terrain of classical stats models, my mentor proposed it would be a good idea to create a general function to which we could apply all of our models. That was the direction we went and it proved to be fruitful, and just as exciting as I had hoped.

I became aware of many aspects in the modeling process such as the importance of particular data structures within a function that in this case, appeared to do the majority of the work. In this case, our rolling_cross_validation method did the majority of the work. This particular method breaks down a dataframe ticker by ticker and populates a sample column based on time index conditions so we can split our data for training, testing and validation. The structure of the method was designed to not only split the data but implement any chosen model by training on a portion of the set and predict one point at a time using a rolling window. This process continues until all future points are predicted, one at a time.

To be more clear, the training data will also grow one point at a time and that new point will be incorporated in the prediction method. This a great idea and will certainly be a method to use in the future.

The method is very easy to understand and manipulate. If I encountered an error in its implementation, it was an opportunity to become acquainted with its structure and modify it according to the needs of the model that was causing the error or once again become acquainted with the use of time indexing.

Often, if I encountered an error it was due to a missing parameter in the call to a model or nested method. Sometimes models require a simple change such as data type. For instance, converting the data from a dataframe to numpy values may be necessary.

At the same time, this whole process of writing and modifying a function has become a familiar habit. The question now is how the function will work best. I was able to see how powerful scripting python functions can be. There is still much to be explored but I have certainly experienced a large portion of the process.

To give you an idea of the type of models used, among them were auto arima, fb prophet, simple average, tbats, auto arima with fourier transforms, and lstm.

I would like to take a moment to discuss more thoroughly the architecture of the LSTM by far the more advanced type of model, which has come to be known as a deep learning model. This particular model is a particular type of neural network. Now neural networks deserve a moment for discussion. They are described in diagrams as graphical models with nodes and edges where the nodes are representative of the input variables and the edges represent weights like that of the weights in a linear regression.

The difference between ordinary regression and neural nets is not just the manifold of variant architectures but the ability to exploit combinations and relationships between all the input signals in the process of estimating an optimal set of weights. Optimization techniques are generally similar in nature and can be reduced to the use of gradient descent in backpropagation. So it is quite an improvement as a means of reducing errors. The LSTM is known as a recurrent neural network and as such it is a sequential based model. Outputs get fed back into the network and the architecture has gates that control the flow of information. As the inputs get processed through hidden states, the keep and forget gates do the majority of the work in remembering relevant information before reaching the output layer. Updates occur in the cell state where all the gates get triggered. And the updates affect the weights with the following calculation:

$$\text{new weight} = \text{weight} - \text{learning rate}*\text{gradient}$$

Using a 3 month data set from a 30 year data set for the purpose of wait time, we were able to see the difference in performance and yet still not the full capability of the models. But among all of the models, auto arima performed the best with a MAPE of 1.49.

The worst performing model was the fb prophet model with a MAPE of 36.9. This could be due to the use of the rolling one step prediction approach.

LSTM took the least amount of time to implement but it is one of the more complicated models in terms of setting up. To prepare a dataset for LSTM modelling, you need to normalize values using a min max scaler and reshape the values, split the data and make sure the values retain their shape in their corresponding train and test object. Following that you have to instantiate several transformation layers before the model receives its input. Then you specify the parameters of the model such as the loss function, optimizer and metrics. Then you compile and fit. It's quite a long procedure. But with the right amount of data LSTM will perform well. But in this case, with just 3 months of data, it received a MAPE of .8.