BT2101 Machine Learning Model Write-Up

LSTM MODEL

Group 23

Shannon Kok Hui Yi A0172090M
Tan Zheng Wei Nicholas A0173054L
Quah Jing Wen A0171964W

1 Summary

In our machine learning model write-up, we will detail the steps that we have taken with respect to the four main aspects: data munging, generation of our LSTM model, use of performance measurements and risk management, along with our thought processes and intentions behind these decisions.

Contents

2 Data Munging

2.1 EXPLORATORY DATA ANALYSIS

2.1.1 High Variance

During data exploration, we found that the high, low, open and close values of each instance differs greatly from one another during a period of an hour, hence there is high variance in the dataset. In order to reduce variance and increase predictive accuracy, we decided to introduce a new feature – mid, which is the mid value of high and low attributes, in our dataset.

2.1.2 Correlation

To diversify our portfolio, we decided to not just focus on 1 currency pair, but a few in relation to a base currency pair. With that, we decided to use correlation to our benefit – by identifying pairs that are positively correlated. An increase in the price of the base currency pair would very likely lead to an increase in a currency pair that is positively correlated to the former. Understandably, this exposes us to directional risk, which we have mitigated by having a Stop Loss and Take Profit (SL-TP) condition that will cut our losses at a lower threshold than our profit, ensuring that we have a higher chance of obtaining a net profit. Thus, we have decided on taking the same actions done on the base currency pair to the selected currency pairs. By doing so, we can minimize losses and gain more profits at the same time.

2.1.3 Data Transformation

Since the values of the dataset are large, we choose to normalize the data to fit it properly into our model. In order to fit our data properly into our model, we performed differencing on our dataset to remove trend and seasonality characteristics, producing a stationary time series dataset.

3 LSTM Model

3.1 INITIATION

With reference to Figure 1, found in the Initalize function, the variables here are used to track the length of operation as well as for data storage, for ease of reference throughout the algorithm. The first variable **self.model** is to store our model that is to be used to generate predictions, while **self.datalist** will store the dataset that will be used should we retrain our LSTM model. We have a retraining factor **self.x** that will train a new LSTM model using the current dataset should it be equal to '0'. The operation duration for our current LSTM model is tracked with **self.count**, and the losses incurred by the current model are tracked with **self.currLoss**. The variables **self.currPeak** and **self.prevPV** track

the current model's best performance as well as previous model's best performance, hence remember to set it as the starting amount of cash that your strategy is intended to have, i.e. 100000 for this model.

```
self.long_list =[]
self.short_list =[]
self.model =Sequential()
self.datalist = np.array([])
self.x=0

#retraining variables
self.count = 0
self.currPeak = 100000 #curr portfolio value
self.prevPV = 100000 #previous portfolio value
self.currLoss = 0 #accumulated loss
self.once = 0
```

Fig 1. Initialisation of Retraining Variables

3.2 MODEL TRAINING

3.2.1 Cells and Epochs

In Figure 2, we first select the number of cells and epochs that we wish to iterate through in our grid search, to find the optimal combination that produces the best accuracy.

```
if self.x==0:  #To make sure the model is build only once and avoid computation at every new data

    # self.Debug("RETRAINING!!!!")

    #USE TimeSeriesSplit to split data into n sequential splits
    tscv = TimeSeriesSplit(n_splits=2)

    # Make cells and epochs to be used in grid search.
    cells = [200,400]
    epochs  = [200,400]

    # creating a datframe to store final results of cross validation for different combination of cells and epochs
    df = pd.DataFrame(columns= ['cells','epoch','mse'])
```

Fig. **2** Selection of cell and epoch values

3.2.2 Grid Search

In Figure 3, 4 and 5, we begin to do the grid search, creating a model for each combination, with which we test and save the results to a data structure. In Figure 3, we first split the dataset that we have into training and test set.

```
for i in cells:
    for j in epochs:

        cvscores = []
        # to store CV results
        #Run the LSTM in loop for every combination of cells an epochs and every
        for train_index, test_index in tscv.split(X_data):
            #self.Debug("TRAIN:", train_index, "TEST:", test_index)
            X_train, X_test = X_data[train_index], X_data[test_index]
            Y_train, Y_test = Y_data[train_index], Y_data[test_index]
```

Fig. 3 Grid search for optimal combination

3.2.3 Building of Model

The process of building the LSTM model for each combination is documented in Fig. 4, where we add the different layers to the model, and record the MSE for each model. It is important to reset the self.model for each iteration, or the layers will just be added to the previous model.

```
model = Sequential()
model.add(LSTM(i, input_shape = (1,3), return_sequences = True))
model.add(Dropout(0.10))
model.add(LSTM(i,return_sequences = True))
model.add(LSTM(i))
model.add(Dropout(0.10))
model.add(Dense(1))
model.compile(loss= 'mean_squared_error',optimizer = 'rmsprop', metrics = ['mean_squared_error'])
model.fit(X_train,Y_train,epochs=j,verbose=0)
```

Fig. 4 Adding of layers to the model

3.2.4 Optimised Model

Here in Figure 5, we find the combination that gave the smallest Mean-squared Error (MSE), and record it in the variables O_cells, O_epochs. Here on, just use these values to train the final model that will be used to generate the prediction.

```
#Check the optimised values obtained from cross validation
#This code gives the row which has minimum mse and store the values to O_values
O_values = df[df['mse']==df['mse'].min()]

# Extract the optimised  values of cells and epochs from above row (having min mse )
O_cells = O_values.iloc[0][0]
O_epochs = O_values.iloc[0][1]
```

Fig. 5 Optimal cell and epoch values are chosen

4 Performance Measurements Used Appropriately

4.1 CONFUSION MATRIX

We did not consider the accuracy of our model using a confusion matrix but applied reinforcement learning based on the accuracy of prediction from the model. If our model is predicting in the right direction, we will reinforce this good performance by adding 10 instances of the previous period's data to the data set. We also analysed our model's accuracy by monitoring the profits it generates.

However, we did use Mean Squared Error (MSE) as a performance metric to sieve out the optimal combination of cells and epochs in the training of our model.

```
92          # if portfolio value greater than currPeak (best performance of current model), reset currPeak
93          elif self.Portfolio.TotalPortfolioValue > self.currPeak:
94              # set the currPeak & prevPV value
95              self.currPeak = self.Portfolio.TotalPortfolioValue
96              self.prevPV = self.Portfolio.TotalPortfolioValue
97              # self.count += 1
98              for i in range(L//10):
99                  ## positive reinforcement
100                 self.datalist = np.concatenate([self.datalist, add_data], axis=1)
101                 self.x= 1
```

Fig. 6 Reinforcement Learning

4.2 AVOIDING OVERFITTING

4.2.1 Conversion of Data to Stationary Data

By making our data stationary, we have ensured that the mean and variance remain constant. This helps us identify the true driving factors of change in our time series, which may be "contaminated" by noise in the non-stationary data. Hence this process ensures that the model does not only do well in the training set, avoiding overfitting situations and improving predictive accuracy.

4.2.2 Ensemble Models

Our group considered using ensemble models, but this will result in large datasets, which is undesirable for our algorithm which is already dealing with large amount of data. This will cause our model to be computationally inefficient and would result in runtime errors. Hence, we decided to not use this method and instead improve our value of data to improve our predictive accuracy.

4.3 RETRAINING

We have chosen 4 conditions to determine whether we should retrain our dataset with the occurrence of each period, and chose our portfolio value to be our attribute of comparison. Since our portfolio value reflects the significance of our holdings, we have to ensure that our model gets trained appropriately such that our portfolio value does not worsen.

5 Risk Management

5.1 BUY/SELL EXECUTION CONDITIONS

Our model does not entirely take actions based on the direction of the prediction, but uses the quantitative figures from the predictions. Our model only has two kinds of actions - long position, position position. To take the short position, the prediction must be at most 90% below the current 'mid' price. We have decided to withhold 10% of our holdings and play with the remaining 90% as a means of preventing too much losses. The amount of

holdings we will be playing with is split among the 3 currency Pairs we invest in with 50%, 30% and 20% weights respectively. (Fig. 7)

```
## if output == "DOWN"
if 0.9*output < price  and self.currency not in self.long_list and self.currency not in self.long_list:
#self.Debug("output is lesser")
#Buy the currency with X% of holding in this case 90%x
    self.SetHoldings(self.currency, -0.9*0.5)
    self.SetHoldings(self.addcurrency1, -0.9*0.3)
    self.SetHoldings(self.addcurrency2, -0.9*0.2)
    self.short_list.append(self.currency)
    self.Debug("short")
```

Fig. 7 Deciding on when to take short position

5.2 STOP LOSS TAKE PROFIT (SLTP) CONDITIONS

5.2.1 Short Position

To prevent from incurring too much losses, we will immediately sell the currency pair when it increases by 0.5%. However, if the currency continues falling in our favour, we will take the profits when the currency dropped by 1%, translating into 1% profit for us. (Fig. 8) The risk:reward ratio we are using is 1:2. Our group used a rather tight threshold on the losses as it is representative of the accuracy of our model. Moreover in the short position, there is no limit to the losses that can be incurred, making it extra important for us to have a stricter threshold. (Fig. 9)

```
# if self.currency in self.short_list:
#     cost_basis = self.Portfolio[self.currency].AveragePrice
for currency in self.short_list:
    cost_basis = self.Portfolio[currency].AveragePrice

#self.Debug("cost basis is " +str(cost_basis))
    #if curr price < average price by 1% or if curr price > 0.5% of average pric
    if  ((price <= float(0.99) * float(cost_basis)) or (price >= float(1.005) * float(cost_basis))):
    #self.Debug("SL-TP reached")
    #self.Debug("price is" + str(price))
    #If true then buy back
        self.SetHoldings(currency, 0)
        self.short_list.remove(currency)
```

Fig. 8 Deciding on when to sell currency in the short position

Fig. 9 Maximum losses from taking long and short positions

5.2.2 Long Position

To take the long position, the prediction must be 5% more than the current 'mid' price.(Fig. 10)

```
# if output == "UP"
# if output X% higher than price, in this case 5%
if 1.05*output > price  and self.currency not in self.long_list and self.currency not in self.short_list :
    self.Debug("output is greater")
    # Buy the currency with X% of holdings, in this case 90%
    self.SetHoldings(self.currency, 0.9*0.5)
    self.SetHoldings(self.addcurrency1, 0.9*0.3)
    self.SetHoldings(self.addcurrency2, 0.9*0.2)
    self.long_list.append(self.currency)
    self.Debug("long")
```

Fig. 10 Deciding when to take the long position

Likewise, to prevent ourselves from incurring too much losses, we will sell the currency pair in the long position if it drops by 0.5%. On the other hand, if the actual value is in line with the prediction from our model and earns us a 1% profit, we will sell the currency pair and keep the profit. The risk:reward ratio used for the long position is same as that of the short position. (Fig. 11)

```
    for currency in self.long_list:
        cost_basis = self.Portfolio[currency].AveragePrice
        #self.Debug("cost basis is " +str(cost_basis))
        # if 0.5% loss -> stop loss, if 1% profit, we will take it
        if  ((price <= float(0.995) * float(cost_basis)) or (price >= float(1.01) * float(cost_basis))):
            #self.Debug("SL-TP reached")
            #self.Debug("price is" + str(price))
            #If true then sell the currencies in long positive
            self.SetHoldings(currency, 0)
            self.long_list.remove(currency)
```

Fig. 11 Deciding on when to sell the currencies in the long position

6 Reference List

1. Shekhar, A. (2018, February 15) What is Feature Engineering for Machine Learning? Retrieved from: https://medium.com/mindorks/what-is-feature-engineering-for-machine-learning-d8ba3158d97a
2. Machado, G. (2016, October 6) ML Basics: supervised, unsupervised and reinforcement learning. Retrieved from: https://medium.com/@machadogj/ml-basics-supervised-unsupervised-and-reinforcement-learning-b18108487c5a
3. Brenyah, B. (2018, January 13) Setting up a Bollinger Band with Python. Retrieved from https://medium.com/python-data/setting-up-a-bollinger-band-with-python-28941e2fa300
4. Oanda (2018, November 1) Currensee Correlation. Retrieved from: https://www.oanda.com/forex-trading/analysis/currency-correlation