Cryptocurrency-predicting

Lokesh Soni

20 October 2018

1. Definition

1.1. Project Overview

A digital currency (or cryptographic money) is a computerized resource intended to function as a medium of trade that utilizations solid cryptography to anchor monetary exchanges, control the production of extra units, and check the exchange of benefits.

Cryptographic forms of money are a sort of elective cash and computerized cash (of which virtual cash is a subset). Cryptographic forms of money utilize decentralized control instead of incorporated computerized cash and focal keeping money frameworks. The decentralized control of every digital currency works through appropriated record innovation, normally a blockchain, that fills in as an open money related exchange database.

1.2. Datasets

Input data fields will be as follows-

- Open and Close
- High and Low
- Volume data
- Ethereum, Litecoin and Bitcoin Cash.

I just centered around the Close and Volume columns.The Close section estimates the last cost toward the finish of every interval. The Volume section is the amount of the advantage was exchanged per every interval, for this situation, per 1 minute. In the least difficult terms conceivable, Close is the cost of the thing and Volume is the amount of thing.

1.3. Problem Statement

This problem is a type of classification problem. Inputs are 60 feature sets of combine price and volume for each coin and the output will be the prediction of the “price” of coin.
The overall goal of the project is to construct a Recurrent Neural Network(RNN) model that can predict price trends with results superior to that of random selection. As the cryptocurrency has a reputation of being a very speculative investment, driven to a high a degree by the emotional frenzy of amateur investors, I will attempt to discern what impact this has on the market by incorporating various different features such as open and closing of the price, High and low in pricing also Volume of particular Coin. I worked only on close and volume data of the all four currencies and using only the last 10% for the input to the model so model can predict future more efficiently as the price of the cryptocurrencies are more fluctuating. For the model architecture I used Recurrent Neural Network(RNN) as RNN works well in sequential data.

1.4. Matrices

The log loss between the predicted values and the original values calculates the prediction results. The loss function which works by penalising the false classifications is known as Logarithmic Loss or Log Loss. For multi-class classification, it works well. While working with Log Loss, which measures the performance of a classification model where the prediction input is a probability value between 0 and 1, the probability to every class for all the samples must be assigned. If there are M classes consisting of N samples, then the Log Loss is calculated as below :

$$LogarithmicLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

Where,

yij	=	indicates whether sample i belongs to class j or not
pij	=	indicates the probability of sample i belonging to class j

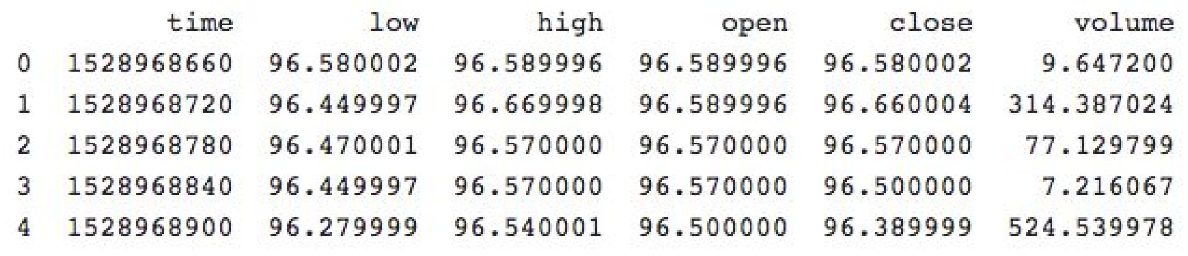
There is no upper bound for Log Loss and its range is [0, ∞). A perfect model would have a log loss of 0, whereas it increases as the predicted probability diverges from the actual label which indicates lower accuracy.

In simple terms, lesser the Log Loss, greater the accuracy for the classifier.

2. Analysis

2.1. Data Exploration

Input data fields will be as follows- Head of LTC-USD



```
         time        low       high       open      close      volume
0  1528968660  96.580002  96.589996  96.589996  96.580002    9.647200
1  1528968720  96.449997  96.669998  96.589996  96.660004  314.387024
2  1528968780  96.470001  96.570000  96.570000  96.570000   77.129799
3  1528968840  96.449997  96.570000  96.570000  96.500000    7.216067
4  1528968900  96.279999  96.540001  96.500000  96.389999  524.539978
```

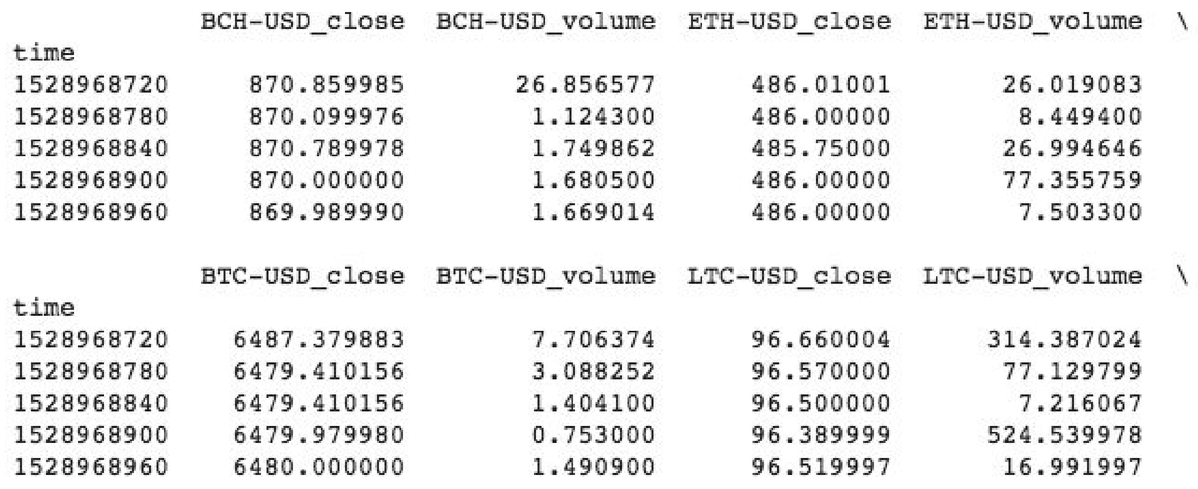
For training the model, I just centered around the Close and Volume columns.The Close section estimates the last cost toward the finish of every interval. The Volume section is the amount of the advantage was exchanged per every interval, for this situation, per 1 minute.

I first took the close and volume from one coin data, and joined it with the other 3 cryptocurrencies. Next thing I did was balance and normalize this data. By balancing the data, I ensure that the classes have equal amount when training the model, so our model doesn't simply predict one class always .Then, splitting the data back to sequence sets and targets .

2.2. Exploratory Visualization

After data exploration, I merged close and volume column of all four cryptocurrencies. Because I am only focused on close value and volume value of the currencies. The close column of all four currencies measures the final price at the end of each interval or at the end of each minute the price of the coin. The volume column describes how much of the coin was traded per interval. As We can see from below data that each currency has very different different close and volume values for the particular interval.

This will be difficult for the model to train so for that I done some normalisation and balancing to the datasets.



```
            BCH-USD_close  BCH-USD_volume  ETH-USD_close  ETH-USD_volume  \
time
1528968720     870.859985       26.856577      486.01001       26.019083
1528968780     870.099976        1.124300      486.00000        8.449400
1528968840     870.789978        1.749862      485.75000       26.994646
1528968900     870.000000        1.680500      486.00000       77.355759
1528968960     869.989990        1.669014      486.00000        7.503300

            BTC-USD_close  BTC-USD_volume  LTC-USD_close  LTC-USD_volume  \
time
1528968720    6487.379883        7.706374      96.660004      314.387024
1528968780    6479.410156        3.088252      96.570000       77.129799
1528968840    6479.410156        1.404100      96.500000        7.216067
1528968900    6479.979980        0.753000      96.389999      524.539978
1528968960    6480.000000        1.490900      96.519997       16.991997
```

2.3. Algorithms and Techniques

Recurrent Neural Nets:

The first algorithm that remembers its input due to an internal memory, is the Recurrent Neural Network which is a class of artificial neural network. This memory of its input makes RNN perfectly suited for Machine Learning problems that involve sequential data. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. It is one of the algorithms behind the scenes of the amazing achievements of Deep Learning in the past few years.

In price predictions problem, inputs (and outputs) are most linked to one another whereas in neural networks such as feed-forward neural nets, they assume all inputs to be independent of each other. So to overcome this problem RNN is used as it shares the information between steps by adding the output of a step to the input of the next one which allows RNNs to have a memory, thus enabling them to distinguish patterns over time.

So for the price prediction of cryptocurrency I need the same functionality, a model that can learn from the past network with backpropagation. RNN model stood out for the particular problem where we have to predict the price and all the input rely on the past output of the network.

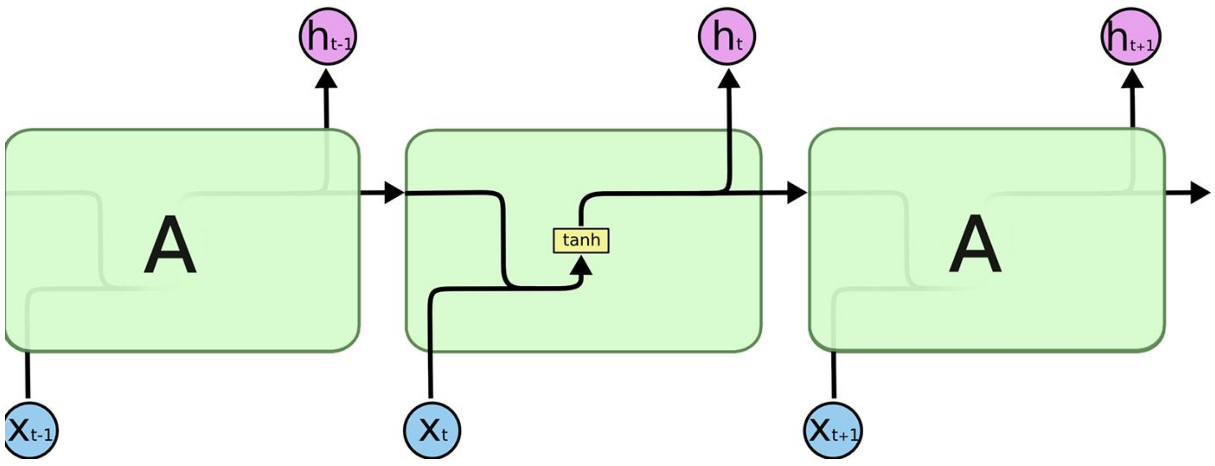


Fig 1. An unrolled RNN, showing the information sharing between steps

Long Short Term Memory (LSTM) :

Long short-term memory (LSTM) units are units of a recurrent neural network (RNN). An RNN composed of LSTM units is often called an *LSTM network*. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications. I used cuDNN LSTM Layers for the building my model architecture. As LSTM model is well suited to time series data or any data with temporal/spatial/structural order. This was the main reason behind the choosing this particular model LSTM out perform other model on the sequential data and our data(Cryptocurrency price data) is sequential. Our LSTM model will use previous data to predict the next one minute's closing price of a specific coin.

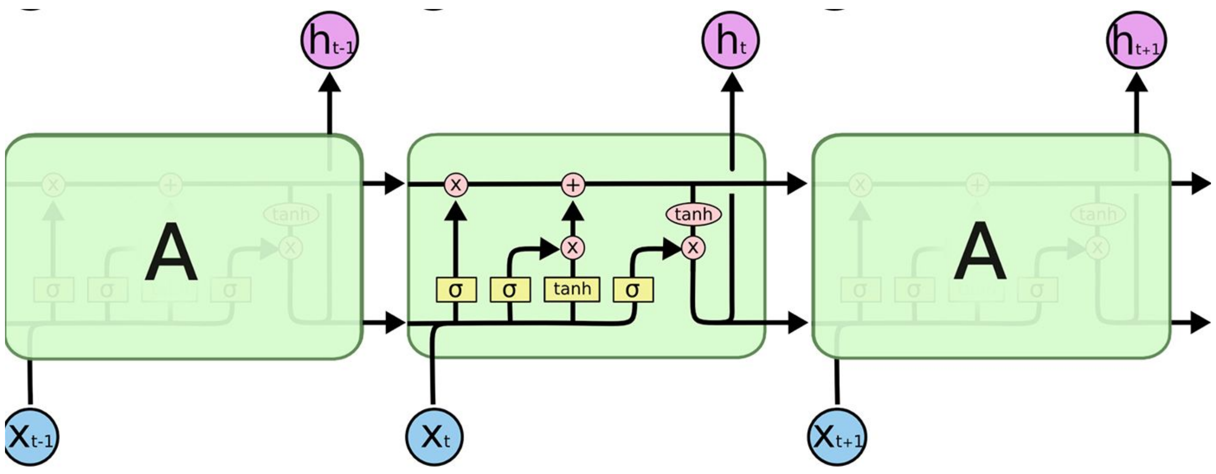


Fig 2. An unrolled LSTM. The unique structure of the LSTM cell allows the gradient to be preserved

2.4. Benchmark

A RNN made from scratch is used as a benchmark model for measuring the performance.

An initial **benchmark RNN** model architecture was defined and then trained on the pre-processed training data. This model has two RNN layers and two dense layers. First RNN layer consist of 128 nodes and has a dropout of 0.2. The second RNN layer consist of 64 nodes and has a dropout of 0.1. All RNN layers are followed by batch normalization. The first dense layers consist of 32 nodes with a dropout of 0.2 and activation function is RELU. The last dense layer consist of 2 nodes with Softmax activation function.

After defining the model architecture It was trained on the training set. After training, predictions .Test Accuracy =0.5792746113680804 , Test Loss =0.6756308129414376 .

3. Methodology

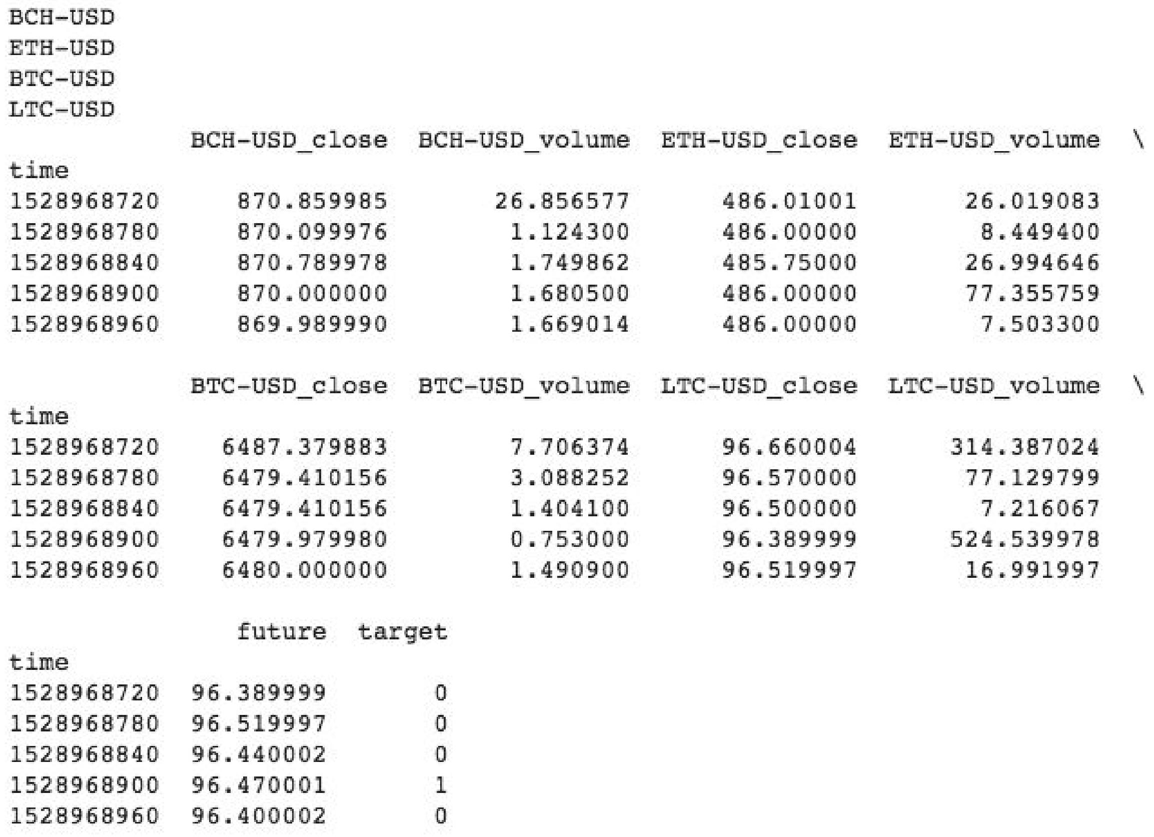
3.1. Data Processing

Separating out our validation of sample data was the first thing I did.

The issue was the data is consecutive, so taking groupings that don't come later on is likely a mistake. The reason behind is the sequences in my case, for instance, 1 minute separated, will be relatively indistinguishable. The objective might also be equivalent (buy or sell) by chances. As a result of this, any overfitting is probably going to really empty over into the validation set. Rather, I slice my validation while it's still all together. I took the last 10% of the data.

Second, I aimed at balancing and normalizing this data. By balancing, I ensure the classes are equal in quantity while training the model, so our model doesn't predict one class always.

For making sequences from it I made a function that will process the data frames, also, I scaled data between 0 and 1. After that I used classify function for classifying buys and sells(or don’t buys) for creating future and target column. If there is raise in price then that is a buys else it is a sell.



```
BCH-USD
ETH-USD
BTC-USD
LTC-USD
            BCH-USD_close  BCH-USD_volume  ETH-USD_close  ETH-USD_volume  \
time
1528968720     870.859985       26.856577      486.01001       26.019083
1528968780     870.099976        1.124300      486.00000        8.449400
1528968840     870.789978        1.749862      485.75000       26.994646
1528968900     870.000000        1.680500      486.00000       77.355759
1528968960     869.989990        1.669014      486.00000        7.503300

            BTC-USD_close  BTC-USD_volume  LTC-USD_close  LTC-USD_volume  \
time
1528968720    6487.379883        7.706374      96.660004      314.387024
1528968780    6479.410156        3.088252      96.570000       77.129799
1528968840    6479.410156        1.404100      96.500000        7.216067
1528968900    6479.979980        0.753000      96.389999      524.539978
1528968960    6480.000000        1.490900      96.519997       16.991997

               future  target
time
1528968720  96.389999       0
1528968780  96.519997       0
1528968840  96.440002       0
1528968900  96.470001       1
1528968960  96.400002       0
```

3.2. Implementation

1. In **model 1** a RNN made from scratch is used as a benchmark model for measuring the performance. An initial **benchmark RNN** model architecture was defined and then trained on the pre-processed training data. This model has two RNN layers and two dense layers. First RNN layer consist of 128 nodes and has a dropout of 0.2. The second RNN layer consist of 64 nodes and has a dropout of 0.1. All RNN layers are followed by batch normalization. The first dense layers consist of 32 nodes with a dropout of 0.2 and activation function is RELU. The last dense layer consist of 2 nodes with Softmax activation function.

After defining the model architecture It was trained on the training set. After training, predictions the Test Accuracy = 0.5491677336747759 and Test Loss = 0.6836623864961495.

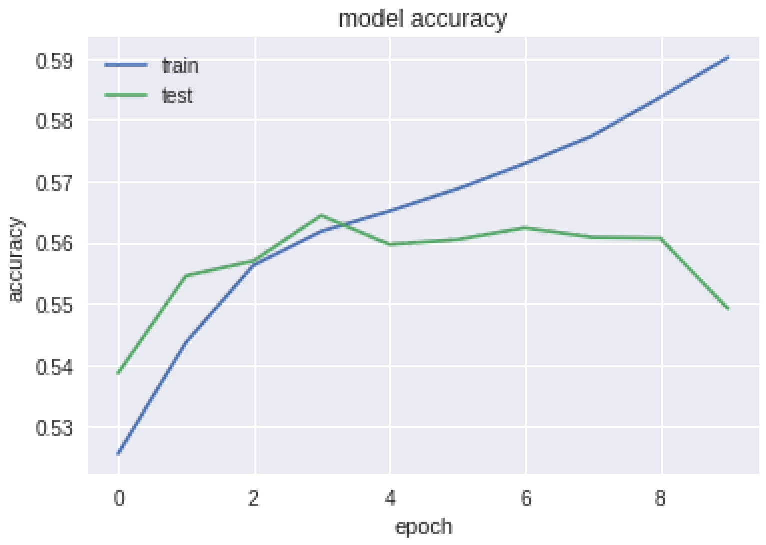


Figure 3 : Model Accuracy for Benchmark Model

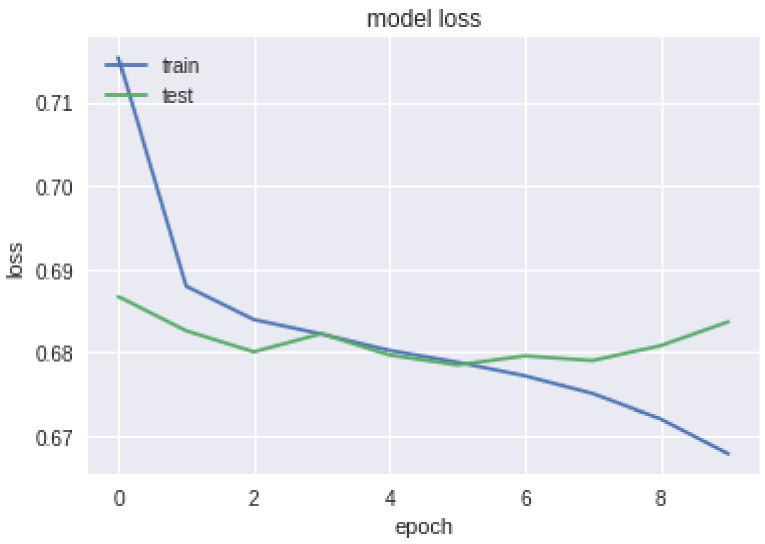


Figure 4 : Model Loss for Benchmark Model

2. The **model 2** has three RNN layers and two dense layers. All 3 RNN layers consist of 64 nodes. First RNN layer has a dropout of 0.2. The second RNN layer has a dropout of 0.1 and third RNN layer has 0.2 dropout. All RNN layers are followed by batch normalization. The first dense layers consist of 32 nodes with a dropout of 0.2 and activation function is RELU. The last dense layer consist of 2 nodes with Softmax activation function.It was trained on the training set. After training, predictions the Test Accuracy = 0.5568501920614597 and Test Loss = 0.6901710304041678.

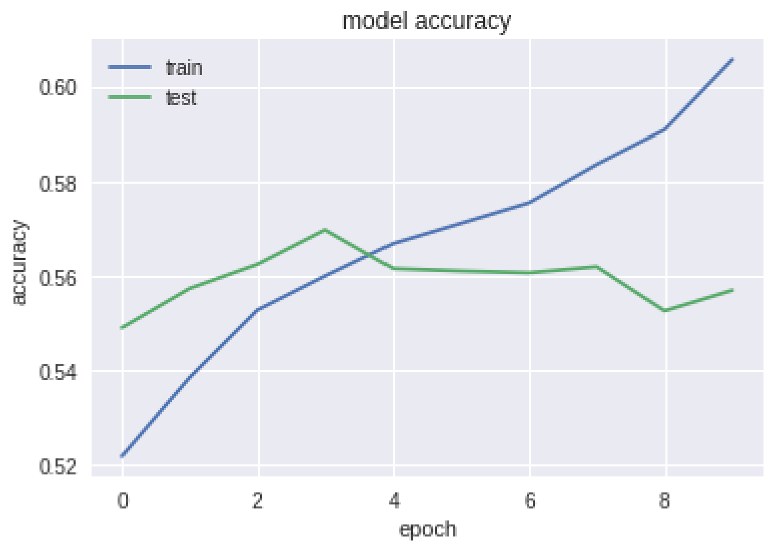


Figure 5 : Model Accuracy for Model 2

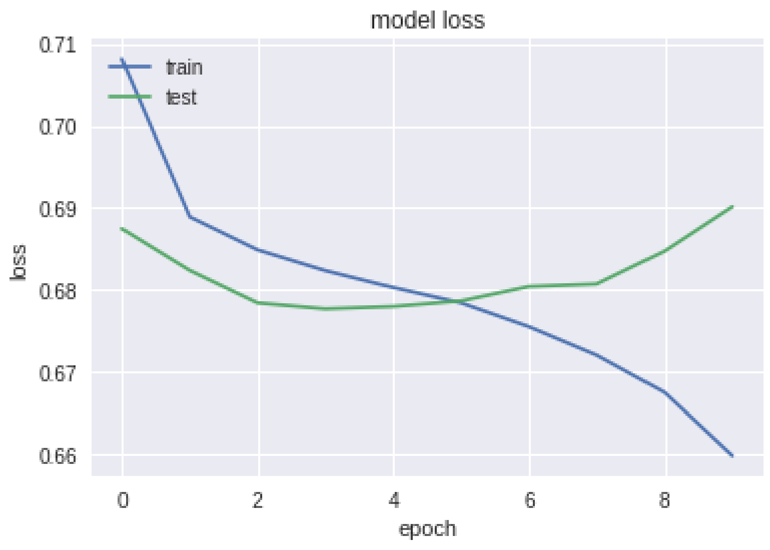


Figure 6 : Model Loss for Model 2

3. The **model 3** has three RNN layers and two dense layers. All 3 RNN layers consist of 128 nodes. First RNN layer has a dropout of 0.2. The second RNN layer has a dropout of 0.1 and third RNN layer has 0.2 dropout. All RNN layers are followed by batch normalization. The first dense layers consist of 32 nodes with a dropout of 0.2 and activation function is RELU. The last dense layer consist of 2 nodes with Softmax activation function. After training this model for 10 epochs, I got a Test Accuracy = 0.506145966709347 and Test Loss = 0.7207571304409208.

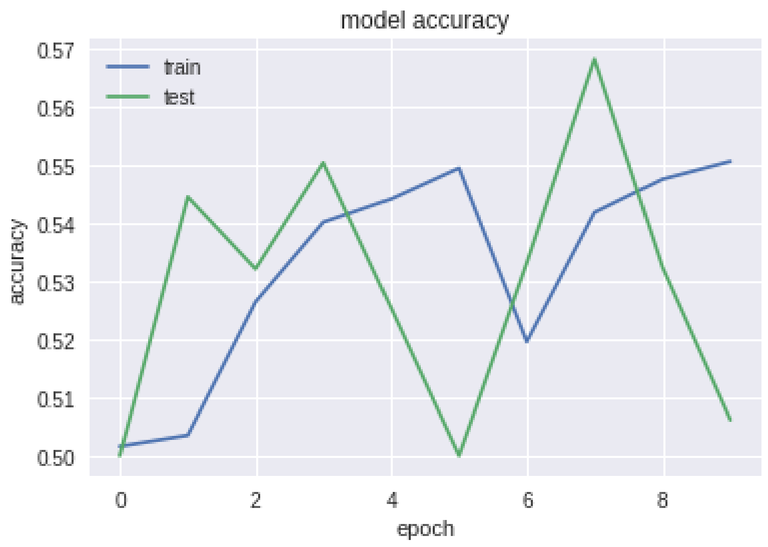


Figure 7 : Model Loss for Model 3

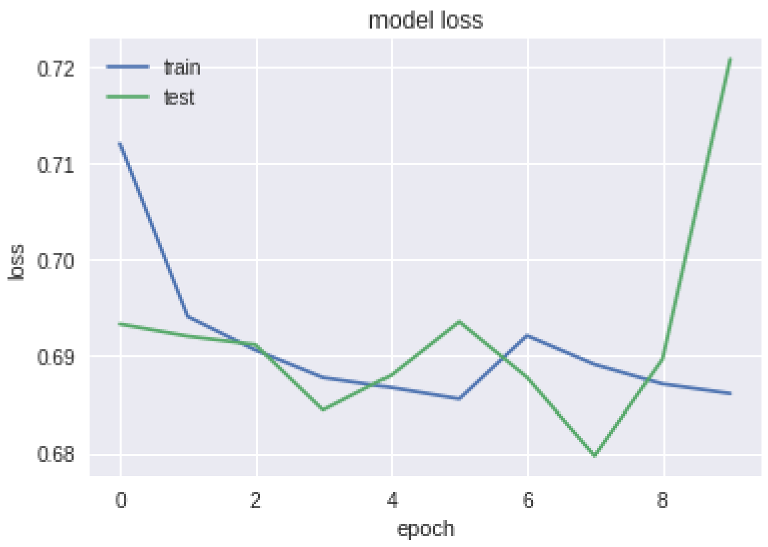


Figure 8 : Model Loss for Model 3

4. The **model 4** has three RNN layers and two dense layers. All 3 RNN layers consist of 32nodes. First RNN layer has a dropout of 0.5. The second RNN layer has a dropout of 0.3 and third RNN layer has 0.3 dropout. All RNN layers are followed by batch normalization. The first dense layers consist of 16 nodes with a dropout of 0.8 and activation function is RELU. The last dense layer consist of 2 nodes with Softmax activation function. After training this model for 10 epochs, I got a Test Accuracy = 0.5624839948783611 and Test Loss = 0.6808163639983202.

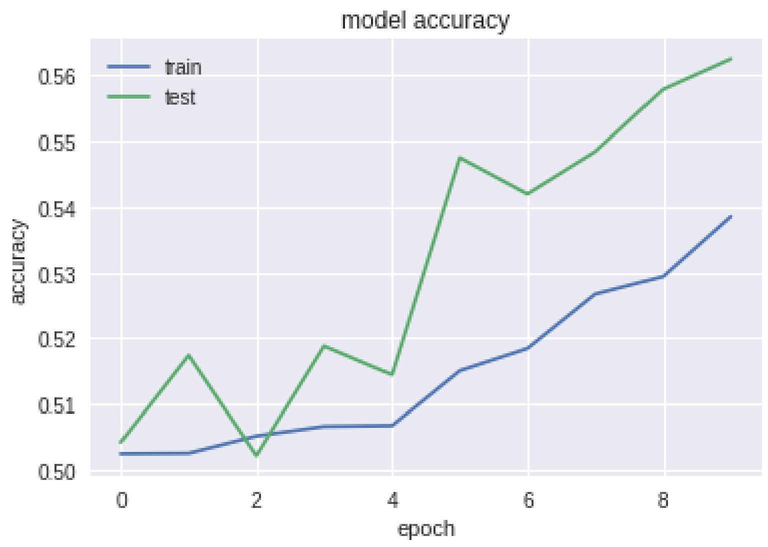


Figure 9 : Model Accuracy for Model 4

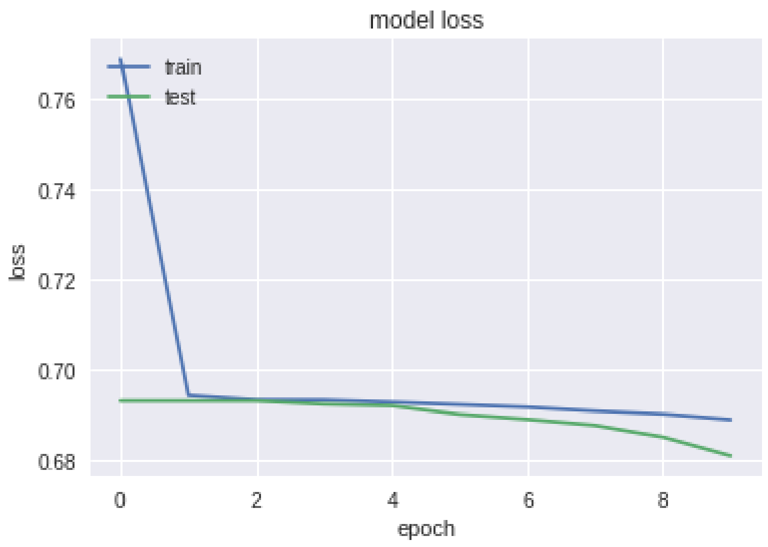


Figure 10 : Model Loss for Model 4

3.3. Refinement

1. **Model 2** :- This model shows a slight increase in accuracy when compared with the benchmark model. I increased from 2 RNN layers to 3 RNN layers with decreasing the number of nodes from 128 in first layer to 64 nodes in all the 3 RNN layers. This model achieved a Test Accuracy = 0.5568501920614597 and Test Loss = 0.6901710304041678 which is better than benchmark model.

2. **Model 3** :- Then, I double the number of nodes in each RNN layer to 128 from 64 in all 3 nodes which decreases the accuracy for training. I train this model for 10 epochs and it achieved a Test Accuracy = 0.506145966709347% and Test Loss = 0.6838349598059382 which is lower than the accuracy of benchmark model.

3. **Model 4** : This model shows a increase in accuracy when compared with the benchmark model. I decreased the number of nodes from 128 to 32 nodes in all the 3 RNN layers with the alteration in the dropout. This model achieved a Test Accuracy = 0.5624839948783611 and Test Loss = 0.6808163639983202 which is better than all other model.

4. Results

4.1. Model Evaluation and Validation

The final model is in the mlnd_capstone_project.ipynb file. On it was capable to achieve accuracy between 50-55% with testing on the dataset. During the project proposal I expected, that the final model architecture will improve accuracy significantly But this model behaved opposite. So I tried with another Model (Model-4). For the final model I choose its parameters by learning from previous models. I decreases the nodes for and increased dropouts for layers this helps for improving accuracy. During the tuning and debugging process, I trained the model several times with training and validation data set. I assume Model-4 is stable and trustable for this dataset but Use at your own risk, Finance is hard.

4.2. Justification

Model 4 performs better than the benchmark and other model as well. Where benchmark model gets a accuracy of 0.5491677336747759, model 4 made a improvement and get a

accuracy of 0.5624839948783611 and also its loss is lower than others followed by model 2 which achieved a accuracy of 0.5568501920614597. Whereas in model 3 accuracy decreases again to 0.506145966709347 which is lower than the benchmark model.
Accuracy of different models used in the project :-

Model	**Accuracy**	**Loss**
Benchmark Model	0.5491677336747759	0.6836623864961495
Model 2	0.5568501920614597	0.6901710304041678
Model 3	0.506145966709347	0.7207571304409208
Model 4	0.5624839948783611	6808163639983202

Table 1 : Accuracy and Loss of each Model

Conclusion

Free-Form Visualization

For training the model, I chose Close and Volume columns. I first took the close and volume from one coin data, and joined it with the other 3 cryptocurrencies. Next thing I did was balance and normalize this data. By balancing the data.Then, splitting the data back to sequence sets and targets. With the training of the data, the final model and the second model performs better than the benchmark and other model as well due to decrease in the number of nodes. Where benchmark model gets a accuracy of 0.5491677336747759, model 4 and model 2 made a improvements and get a accuracy of 0.5624839948783611 and 5568501920614597.

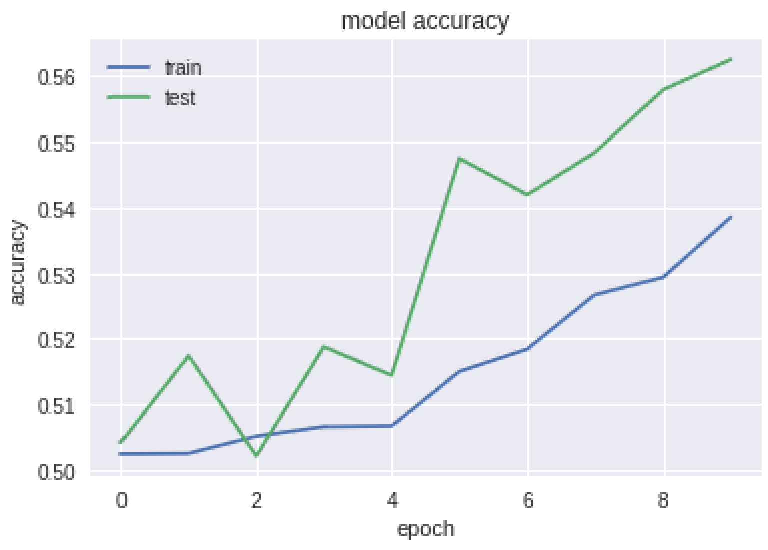


Figure 11 : Model Accuracy for Model 4

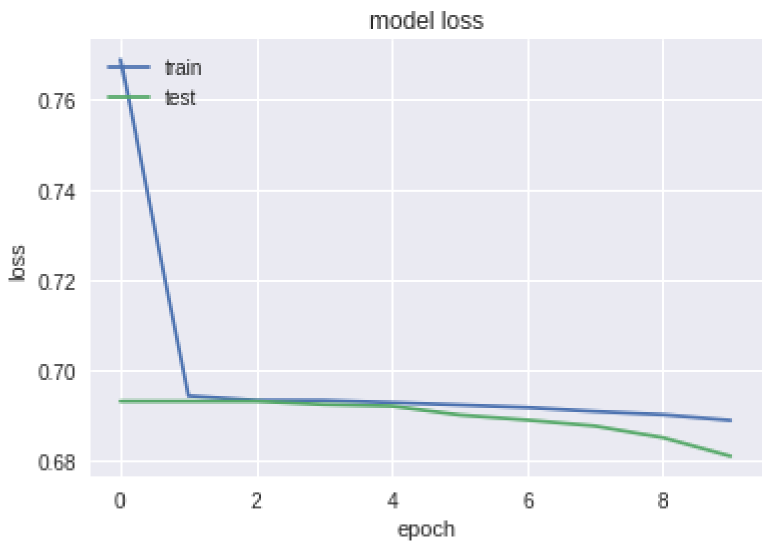


Figure 12 : Model Loss for Model 4

Reflection

In this project, I create a Recurrent neural network model for predicting cryptocurrency. First I built a benchmark architecture from scratch and then try to improve my results by increasing and decreasing the number of nodes for the other models. The second and final machine learning model in which I decrease the number of nodes achieved better results than the benchmark model. The models was trained directly on the data. The RNN based models perform better in the field of classification and therefore the results of the project are as expected. However, in the beginning, I thought that the RNN model will be capable to learn even from unnormalized data. Also I saw that while increasing the number of node I got the accuracy less than that of the benchmark model which was opposite than I expected. I will look forward doing some more projects to learn more about them.

Improvement

The Model 4, could be improved significantly if I decrease more number of nodes along with the alterations in the dropout. The model could be trained for more numbers of epochs and also by using a more powerful machine as this is very time consuming and power consuming. Also we could run some more experiments using various architectures.

References

- https://en.wikipedia.org/wiki/Dropout_(neural_networks)
- https://keras.io/
- https://en.wikipedia.org/wiki/Cryptocurrency
- https://en.wikipedia.org/wiki/Long_short-term_memory
- https://en.wikipedia.org/wiki/Recurrent_neural_network
- https://pythonprogramming.net/static/downloads/machine-learning-data/crypto_data.zip
- https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5
- https://www.tensorflow.org/
- https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234