Bitcoin Prediction using LSTM

Raphaël CANIN, Melisa KOCKAN
E4 AIC

Introduction

The world of cryptocurrencies is still very young. With the lack of real regulations, the volatility on this kind of market is sometimes extreme. One can become rich or lose everything in the span of a few days. Just like the more traditional stock market, many algorithms have been imagined and designed to predict the future price and thus be able to make entries that will generate money.

However, a basic rule of data science is not respected here: in order to have a performing model, the problem must be solved by a human expert. However, it is almost impossible to predict the price of crypto-currencies, as fluctuations depend on many different variables (cycles, volumes, news...).

In reality, the market is a zero sum game. For someone to win $20k, someone else has to lose $20k on the other side. The price will thus take the direction where there is the most liquidity to be taken (liquidity grab). For example, if 90% of the people are long BTC (especially with leverage), it will tend to fall to clean up their stop loss or simply liquidate them. So, to predict bitcoin, you actually have to analyze the psychology of the market participants.

Even though the problem is not solvable, we still wanted to look into this topic because Raphael is passionate about the cryptocurrency world. We will try to come up with a coherent solution, even if it will not necessarily work very well.

Dataset

There are thousands of datasets on bitcoin. Only, most of the time, the only features available are the price (in the form of candles) and the volume. As explained above, you should trade people, not price.

There are several psychological indicators for cryptocurrency. Here we will focus on Open Interest. These are calculated based on the number of open contracts in the futures market. Without going into details, it allows to see if the majority of people buy the dip or FOMO (Fear Of Missing Out). To get a dataset containing this indicator, it is necessary to have a premium (expensive) subscription on TradingView. We will thus use Raphaël's account that already have a premium account.

The choice of the timeframe is crucial. Even though the market is fractal, small enough time frames tend to be chaotic. A larger time frame will smooth out the imperfections in our data. The time frame we will choose is 4 hours. This means that each row in our dataset will be spaced 4h apart.

Fig. 1 - TradingView interface

We use the export data button, taking the maximum possible time range (until about July 2021) to obtain a csv file.

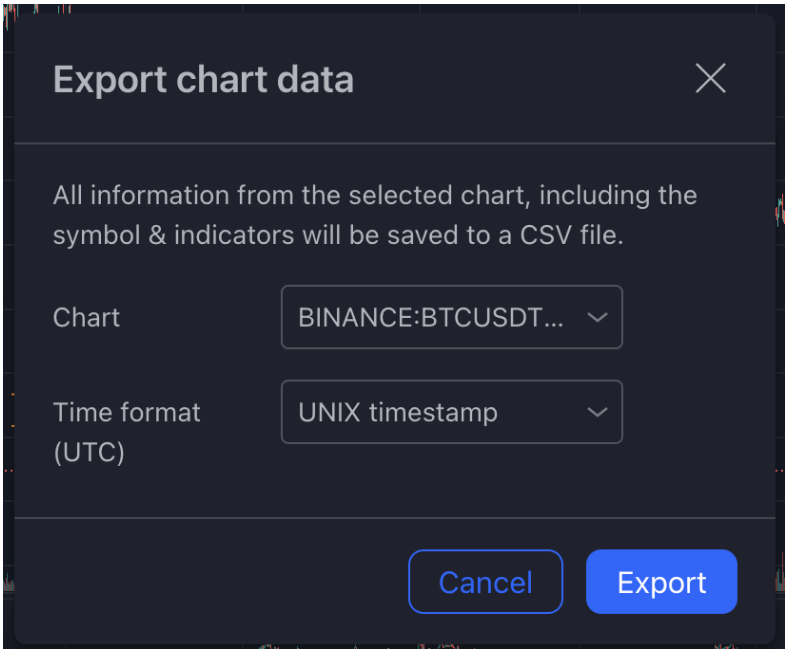

Fig. 2 - CSV Export

Data Visualization

After importing the dataset using the pandas module, we remove all rows containing nan.

```
data = pd.read_csv('BINANCE_BTCUSDTPERP, 240 OI.csv')
```

```
data.dropna(axis=1, how='all', inplace=True)
```

```
data.dropna(inplace=True)
```

```
col = data.columns
print(col)
```

```
Index(['time', 'open', 'high', 'low', 'close', 'Volume', 'Volume MA',
       'Crypto Open Interest (Ouverture)', 'Crypto Open Interest (Haut',
       'Crypto Open Interest (Bas)', 'Crypto Open Interest (Fermeture)'],
      dtype='object')
```

Fig. 3 - Import and drop nan

As you can see, we have many columns :

- time : the time index in UNIX time format. We will not use this column because our model will be sliding window based. We dropped this column.
- open, high, low, close : all the characteristics of a prize candle.
- volume, volume MA : the volume of bitcoin exchanged in a unit of time and its moving average.
- Crypto Open Interest (open, high, low, close) : all the characteristics of an open interest candle.

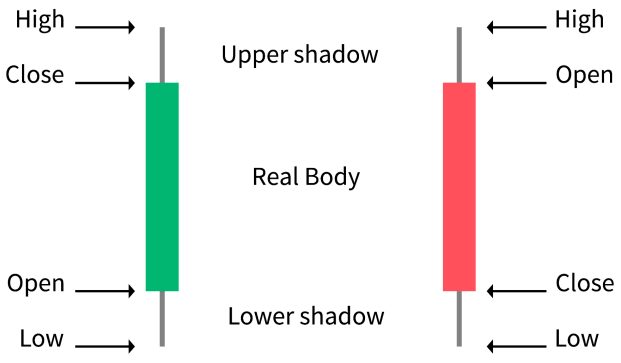

Fig. 4 - How to read candles (source)

As you can see, the dataset is consistent with curves similar to what we had on TradingView. Idem for the OI and Volume.

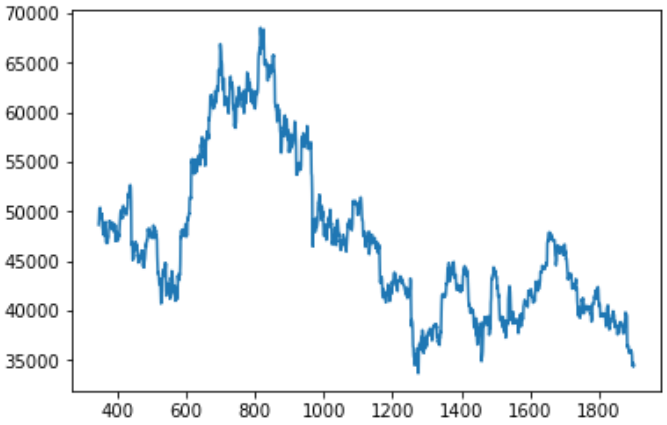

Fig. 5 - Evolution of price in our dataset

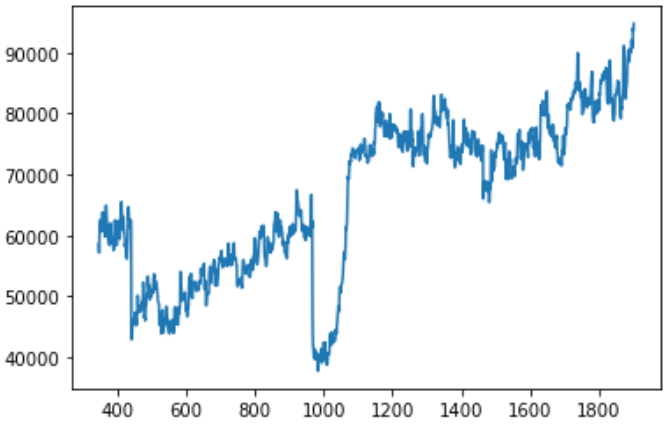

Fig. 6 - Evolution of OI in our dataset

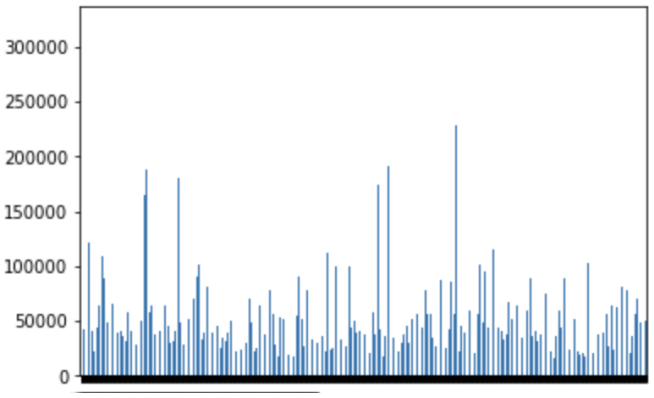

Fig. 7 - Evolution of Volume in our dataset

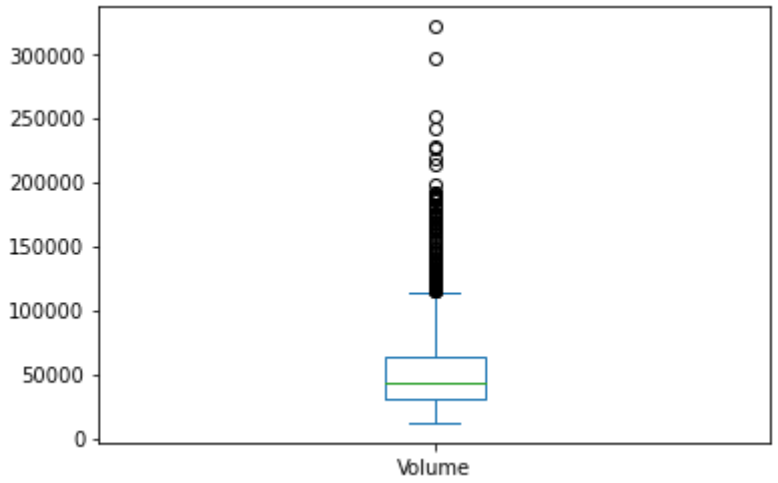

Fig. 8 - Boxplot of volume

Preprocessing

We will now preprocess our dataset. The problem being temporal, we will use a sliding window system of size 10. Of course, we will first normalize the values using a MinMaxScaler from Sklearn.

```
scaler = MinMaxScaler()
scaled = scaler.fit_transform(data)
```

Fig. 9 - Min Max Scaling

A 10-sliding window is a time window containing 10 consecutive rows of the dataset. Thus, our model will evaluate a succession of rows and not only one row. This allows it to have more context on the situation to evaluate. We then divide the dataset into train and test sets.

```
for i in range(window_size, training_data.shape[0]):
    X_train.append(training_data[i-window_size:i])
    Y_train.append(training_data[i])
```

Fig. 10 - Creation of the sliding windows

```
: X_train, Y_train = np.array(X_train), np.array(Y_train)
  print("From", data.shape, "To", X_train.shape)
From (1555, 10) To (1545, 10, 10)
```

Fig. 11 - Evolution of the shape of our dataset

```
X_train, Y_train = X_train[:-window_size*ratio], Y_train[:-window_size*ratio]
X_test, Y_test = X_train[len(X_train)-window_size*ratio:], Y_train[len(Y_train)-window_size*ratio:]
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(1165, 10, 10)
(380, 10, 10)
```

Fig. 12 - Train test split

Model

We are going to use a recursive neural network model in order to manage the temporal sliding windows more easily. For this, we will use the LTSM (Long Short Term Memory), which has feedback junctions.

```
model = Sequential()

model.add(Normalization())

model.add(LSTM(128, activation = 'relu', return_sequences = True, input_shape =

model.add(Dropout(0.2))
model.add(LSTM(256, activation = 'relu', return_sequences = True))

model.add(Dropout(0.3))
model.add(LSTM(512, activation = 'relu', return_sequences = True))

model.add(Dropout(0.4))
model.add(LSTM(16, activation = 'relu'))

model.add(Dropout(0.5))
model.add(Dense(len(Y_train[0])))

optimizer = tf.keras.optimizers.Adam(clipnorm=1)


model.compile(optimizer = optimizer, loss = 'mean_squared_error')
```

Fig. 13 - Our LSTM model

As you can see, we insert layers of dropouts to avoid overfitting. We use relu activation functions for the hidden layers. The output layer is composed of the same number of neurons as there are columns in our dataset. Indeed, we want to predict a whole row from a group of 10 consecutive rows.

To avoid having a huge loss, we will use an optimizer that limits the norm to 1. We use the mean squared error, relevant for a regression.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 10, 10)           21
 n)

 lstm (LSTM)                 (None, 10, 128)           71168

 dropout (Dropout)           (None, 10, 128)           0

 lstm_1 (LSTM)               (None, 10, 256)           394240

 dropout_1 (Dropout)         (None, 10, 256)           0

 lstm_2 (LSTM)               (None, 10, 512)           1574912

 dropout_2 (Dropout)         (None, 10, 512)           0

 lstm_3 (LSTM)               (None, 16)                33856

 dropout_3 (Dropout)         (None, 16)                0

 dense (Dense)               (None, 10)                170

=================================================================
Total params: 2,074,367
Trainable params: 2,074,346
Non-trainable params: 21
_________________________________________________________________
```

Fig. 14 - Model summary

After training on 40 epochs, we get a correct loss for the train and the test. There does not seem to be any overfitting.

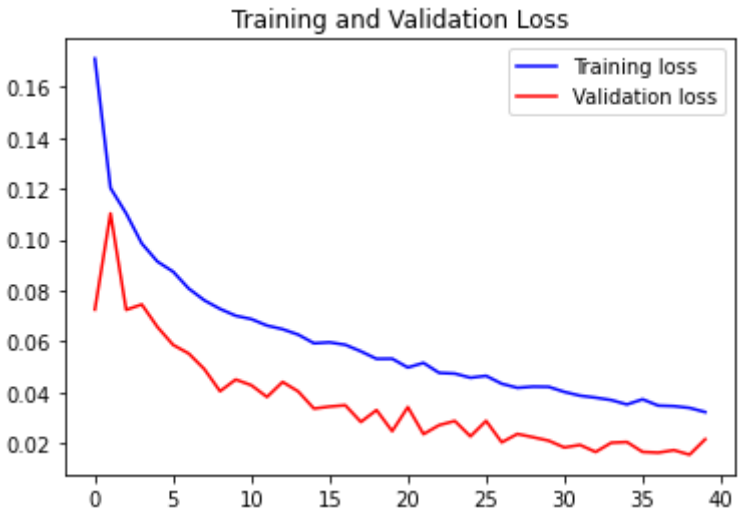

Fig. 15 - Train and Test Losses

Trends Prediction

Since it is very complicated to predict the future price of BTC, we will instead try to predict these trends. Thus, the algorithm will inform us with a certain depth (number of time units in the future) if it will increase or decrease.

So, to do this, we start from a sliding window between t-10 and t and make a prediction on it. We then get a line at t+1. We can then predict again between t-9 and t+1 and so on until t+20.

Then we just have to check if the price evolution between t and t+20 is positive or negative.

By testing this algorithm on our validation set, we get not too bad results.

```
: print(counts[1][1] / len(val) *100, " % of correct trends on", tries, "tries")
```

```
76.6016713091922  % of correct trends on 359 tries
```

Fig. 16 - % of correct trends

Nevertheless, it is not very precise. I'll say it again, predicting cryptocurrency is a very difficult topic. Let's switch on the third subject.