HKUST Business School

ISOM3350 FinTech and Cryptoventures

Bitcoin Low Frequency Trading Strategy Using Price, Trend & Sentiment Analysis

CHENG, Lai Him (20596158)

YEUNG, Man Yin Michael (20603418)

YAU, Tsz Fung (20512970)

SZE, Kai Tik (20496011)

Instructor: Prof. CHEN, Yanzhen

Spring Term, 2021

Table of Contents

Executive Summary

Trading in cryptocurrency has become a hot choice for investors. In the last 12 months, the market capitalization increased fourfold to US$764B, and the value of all altcoins also appreciated from US$60B to 225B by 270%+ [Y. Barabash, 2021]. The increasing awareness motivated us to develop a profit-generating trading strategy for Bitcoin (highest market cap as at April 2021). In our analysis, we first implement three independent innovative models generating daily "scores", which are used to deduce buy/sell trading signals. We consider historical prices (to look for patterns, future predictions and "lookback" signals) and market sentiment analysis using tweets. Some of the results are satisfying, and finally we linearly combine the "score" of these three models as factors to seek an even higher backtest return.

Finalized Models

Part I. Price Prediction using LSTM

Past price behavior and patterns may reflect the human behavior of the investors, and are hypothesized to be useful for future price predictions. Prediction of future prices is useful for the generation of trading signals as profit is generated when future price rises. With the attempt of SARIMAX, XGBoost and Regression Tree models, LSTM model is selected as it demonstrates best predicting power after comparison. LSTM is powerful in sequence prediction with the ability to store past information, showing previous prices are crucial for prediction. Applying linear function as activation function, and adagrad as optimization

method, we minimize train RMSE to 292.42 & test RMSE to 141.05 after 500 iterations. Figures in *Appendix III* demonstrate line charts of dataset and training result, reflecting prediction result matches with historical data. We then calculate "score" for test period to generate trading buy/sell signals by thresholds grid-search. *(Refer to Appendix III for details)*

Part II. Trend-following Strategy with Lookback Signals

We simulate the "Turtle Model" and see if the signals convey insights for trading Bitcoin - a cryptocurrency. Per day, we calculate the rolling maximum and minimum given a lookback, e.g. 60 days, and generate "buy signal" if the current price exceeds the maximum, vice versa. Our idea is that maximum in a lookback period act as a "resistance price" and the minimum as "support price". We hypothesize that resistance-breaking close price indicates a future rise, vice versa. In our model, we adopt an innovative twist to take weighted sums of different lookback signals (1 or -1). Further enhancing, the grid search of parameters is implemented including sets of multiple lookbacks, sets of weights, and threshold pairs to generate buy/sell signals by the weighted sum. We found that 60-day buy signal and 90-day sell signal are most insightful, i.e. profit-generating. We then test if this "chart pattern" is also working in other periods. The test period result shows a similar return p.a. *(Refer to Appendix IV for details)*

Part III. Sentiment Analysis using LSTM

We hypothesize sentiment from tweets as an important factor for the price, as we think most Bitcoin investors are individuals instead of banks or institutions. Thus, the sentiment on

Twitter (a common social media) may reflect the sentiment for these investing individuals. In our model, we first chose VADER to calculate sentiment scores from bitcoin-related tweets as it attuned specifically to sentiments expressed in social media and can give more detailed information. After that, we trained models to predict the future price and buying signal. With the attempt of Decision Tree and TF-IDF, the LSTM model is finally selected as our training model because it demonstrates the best predicting power after comparison. As aforementioned, LSTM can store past information and reflect the past price trend and sentiment information. We could minimize the training mean absolute error (MAE) to 89.09 and the test MAE to 240.15. Using the prediction result, we calculate a score for the test period to generate buy/sell signals reflecting the investment suggestions. *(Refer to Appendix V for details)*

Part IV. A Combination of the Three Models *(Refer to Appendix VI for details)*

We linearly combine the scores output by the three aforementioned model, attempting to attain a cross-validation of the three models for more accurate signals. The result was found to much higher than three 3 models, and having an abnormally high return of 127-fold annually.

Summary of Results & Conclusion

The three separated models are all proven to be profit-generating using the backtests. However the combined model is generating a return even higher than all these three models. We conclude that price, trend and sentiment are useful factors to be considered when trading Bitcoin; and combining factor can enhance the trading strategy in this case.

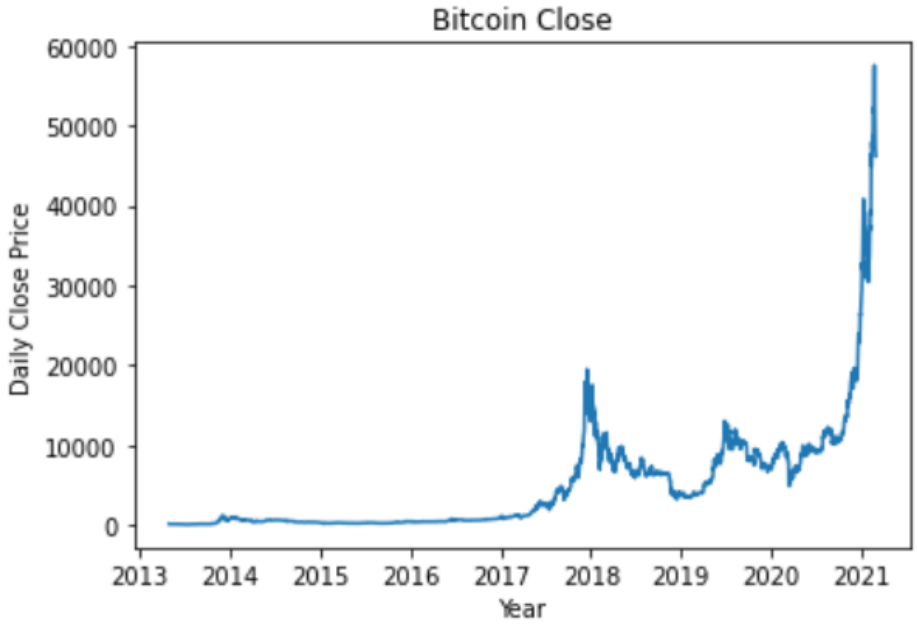
Appendices

Appendix I: Data Collection:

The first dataset includes 4.8M Bitcoin price points from Dec. 2011 to Mar. 2021. The second dataset includes 16M Bitcoin tweets (only 3M used) from May 2013 to Jun. 2019. Both datasets are obtained from Kaggle, and are available using the links in the reference section.

Appendix II: Explorative Data Analysis

Historical Prices of Bitcoin (Close) are very volatile as we can observe in the figure below. As we can see, prices before 2017 are relative very low, thus we do not include those data in our major analyses. Instead, we focus on the period starting from 2017, and ending 2020 (Dec). Different training and testing periods may be used in separated models due to the availability of data (as some parts are using sentiments as well).



We can also observe from the `df.describe()` that the prices are very volatile (high S.D.), and the maximum price ~US$57k is a lot greater than minimum price ~US$2.9k. In our model, we decided to mainly focus on the close prices of this dataset.

	SNo	High	Low	Open	Close	Volume	Marketcap
count	2862.00000	2862.000000	2862.000000	2862.000000	2862.000000	2.862000e+03	2.862000e+03
mean	1431.50000	4974.040239	4695.103027	4836.306834	4852.092547	8.978475e+09	8.591622e+10
std	826.33256	7188.836678	6667.197596	6933.573446	6975.105869	1.658135e+10	1.287414e+11
min	1.00000	74.561096	65.526001	68.504997	68.431000	0.000000e+00	7.784112e+08
25%	716.25000	426.047752	415.675751	421.204506	420.989243	2.786250e+07	5.988997e+09
50%	1431.50000	1197.334961	1164.174988	1180.100037	1182.809998	3.301950e+08	1.924238e+10
75%	2146.75000	8138.046589	7703.357500	7924.612338	7926.696939	1.296743e+10	1.387658e+11
max	2862.00000	58330.572142	55672.609513	57532.738864	57539.943668	3.509679e+11	1.072263e+12

The market sentiment of Bitcoin in tweets is also included in part of our model for analysis. We collected 16M tweets from a dataset in Kaggle. However, as the dataset is very large and training all these 16M tweets requires a super-long runtime, we decided to take a 3M random sample from the tweets for our analysis. This dataset is only use in our sentiment model, details can be found in *Appendix V*.

timestamp	
2019-05-27 11:49:14+00:00	È appena uscito un nuovo video! LES CRYPTOMONN...
2019-05-27 11:49:18+00:00	Cardano: Digitize Currencies; EOS https://t.co...
2019-05-27 11:49:06+00:00	Another Test tweet that wasn't caught in the s...
2019-05-27 11:49:22+00:00	Current Crypto Prices! \n\nBTC: $8721.99 USD\n...
2019-05-27 11:49:23+00:00	Spiv (Nosar Baz): BITCOIN Is An Asset & NO...

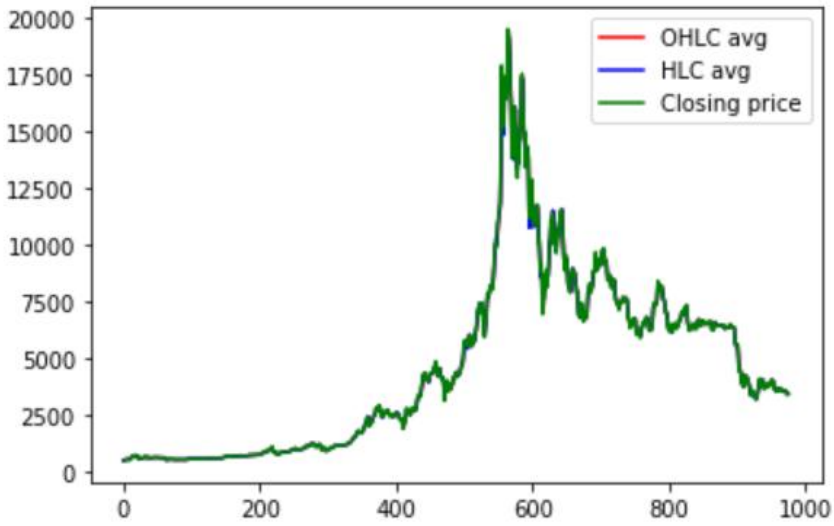
The preprocessing of the text data is done in *part3a_sentiment_data_preprocessing.ipynb* which converts the raw text data into numerical values that are much more convenient for our analysis. Further analysis on the data is done in Part 3(b).

Appendix III: Detailed Explanation of Price Prediction using LSTM

For full codes of this model (Part I), please refer to these files:

part_1_price_prediction_lstm.ipynb *preprocessing.py* *return_calculation.py*

The following paragraphs will briefly introduce some of the main concepts, visualizations, and important parts of the codes. We first preprocess the data and do some cleaning, the cleaned data is visualized as follows:



For the LSTM model we adopt a sequential model in this part, using `keras.models`:

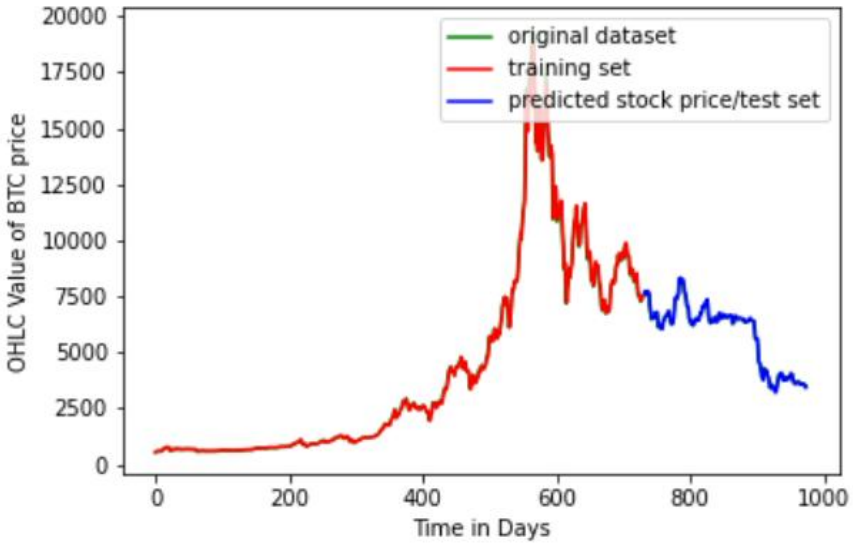
```
# LSTM MODEL
model = Sequential()
model.add(LSTM(32, input_shape=(1, step_size), return_sequences =
True))
model.add(LSTM(16))
model.add(Dense(1))
model.add(Activation('linear'))
```

Training the Keras model:

```
# MODEL COMPILING AND TRAINING
model.compile(loss='mean_squared_error', optimizer='adagrad')
for i in range(501):
    model.fit(trainX, trainY, epochs=1, batch_size=1, verbose=2)
    if i % 100==0:
        model.save('saved_model/model_' + str(i) + '.h5')
```

Making predictions after the model is built:

```
# PREDICTION
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```



The predictions using the past data is visualized as the figure above. After we get the predicted price, we calculate the difference of the predicted value at the specific day with the five following days, and calculate the weighted average with the weightings 5/15, 4/15, 3/15, 2/15, 1/15 respectively as we emphasize on the price change of the nearer future (this can be further improved to be “trained weights”, but we keep them as fixed values for simplicity and saving runtime).

```
for item in range(len(test_predict)-5):
    diff1 = test_predict.loc[item,'test_predict'] -
test_predict.loc[item+1,'test_predict']
    diff2 = test_predict.loc[item,'test_predict'] -
test_predict.loc[item+2,'test_predict']
    diff3 = test_predict.loc[item,'test_predict'] -
test_predict.loc[item+3,'test_predict']
    diff4 = test_predict.loc[item,'test_predict'] -
test_predict.loc[item+4,'test_predict']
    diff5 = test_predict.loc[item,'test_predict'] -
test_predict.loc[item+5,'test_predict']
    test_predict.loc[item,'score'] = 5/15*diff1 + 4/15*diff2 +
3/15*diff3 + 2/15*diff4 + 1/15*diff5
test_predict = test_predict.fillna(0)
```

Finally, we normalize the weighted average into the range between -1 and 1.

```
score = np.reshape(np.array(test_predict.score),
(len(test_predict),1))
scaler = MinMaxScaler(feature_range=(-1, 1))
score = scaler.fit_transform(score)
score = pd.DataFrame(score,columns = ['Score'])
```

If the score (normalized weighted average) is larger and closer to 1, it indicates that buying the asset (Bitcoin) on the corresponding day has a higher probability to get a positive and greater return. We calculate the return of the test period using a pair of threshold. This pair of threshold is obtained using grid search. For the thresholds $\{\tau_B, \tau_S\} \in [-1,1]$, in our model, we assume:

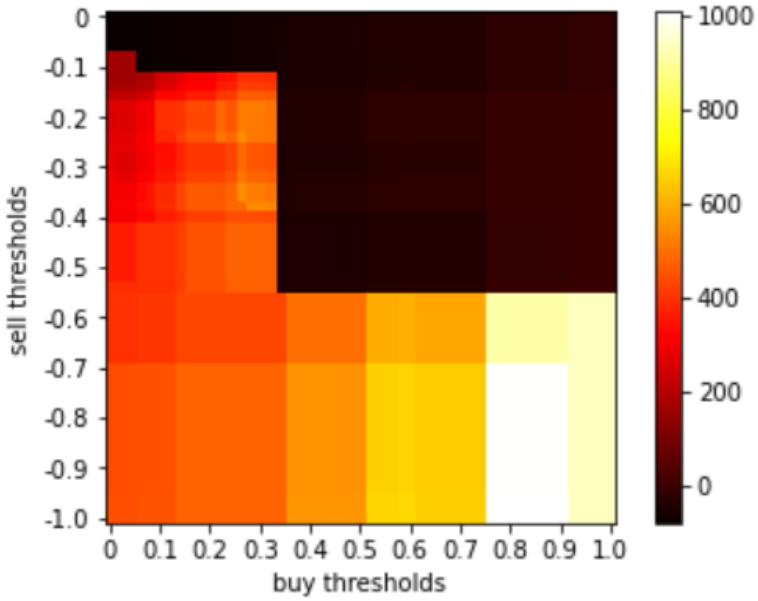
$$(\tau_B, \tau_S) \in \{0.00, 0.02, 0.04, \dots, 1.00\} \times \{0.00, -0.02, -0.04, \dots, -1.00\}$$

```
import itertools
# a list of all possible thresholds (bundle)
thres_list_1 =
[list(np.linspace(0,1,51)),list(-np.linspace(0,1,51))]
thrs_list = list(itertools.product(*thres_list_1))
```

We then apply a Grid Search for the best thresholds by testing the return, i.e. we loop through all the possible thresholds (with a fixed difference) to find the best pair, and saving all returns in a dictionary for generating the color map to be introduced later. This process is done using the following codes:

```
max_ret = 0
count = 1
ret_dict = dict()
for thr in thrs_list:
    percentage = round(count*100/len(thrs_list),2)
    print('Progress: {Percentage}%'.format(Percentage=percentage),
end='\r')
    count += 1
    ret =
rc.get_return(score[['Score']],'Bitcoin',dt.datetime(2013,4,29).date
(),dt.datetime(2021,2,27).date(),thr[0],thr[1])
    ret_dict[thr] = ret
    if ret > max_ret:
        max_ret = ret
        max_thr = thr
```

The result of grid search is visualized as follows:



The color axis indicated the test return (in %). The area with the lightest color indicates the thresholds yielding the highest return. We can see that more polarized thresholds (i.e., thresholds with a greater magnitude) generate a higher return in general. The best pairs of thresholds are located in the white rectangle in the right-bottom corner as shown in the visualization above. The finalized investment returns using the thresholds (0.76, -0.96) output as follows:

```
From 2018-06-04 to 2019-01-31
Return: 1007.18% with parameters t:(0.76, -0.96)
```

We can see from the output above (calculated in *return_calculation.py*) that the return over the test period of 242 days is approximately 1007.18%, which is equivalent to 3638% p.a. (assuming compounding, and 1 year = 365 days). The investment value is expected to be increasing 37-fold per year, which is quite abnormal (however, reflecting behaviors of Bitcoin prices in some periods, much different from prices of other kinds of financial assets).

Appendix IV: Detailed Explanation of Trend-following Strategy

For full codes of this model (Part II), please refer to these files:

part_2_trend_following.ipynb *return_calculation.py*

The following paragraphs will briefly introduce some of the main concepts, visualizations, and important parts of the codes.

An original trend-following trading model was first introduced by Richard Dennis et al. back in 1983 as the "Turtle Experiment", and has been widely recognized as a tremendous success. We simulated this model in our Bitcoin prices backtest, with a twist to adopt a weighted average of different lookback signals. The "score" for a single look-back signal at time t can be mathematically expressed as:

$$\Theta_{i_j t} = \begin{cases} 1 & P_t > \max_{k\in\{1,2,\dots,i_j\}} P_{t-k} & Buy\ Signal \\ 0 & \min_{k\in\{1,2,\dots,i_j\}} P_{t-k} < P_t < \max_{k\in\{1,2,\dots,i_j\}} P_{t-k} & No\ Signal \\ -1 & P_t < \min_{k\in\{1,2,\dots,i_j\}} P_{t-k} & Sell\ Signal \end{cases}$$

where P_t is the price of the asset (Bitcoin) at time t, and

i_j is the period of look-back, for example, 60 (days)

We can see that a single lookback signal is similar to an indication of whether the current price is breaking an "n-day resistance" or "n-day support". In our model, we believe that if we receive an "n-day lookback buy signal" (current close price greater than the n-day rolling maximum), then the price is breaking an "n-day resistance" and thus the price of Bitcoin is going to rise in the future. In this case, the investor should buy the asset, i.e. Bitcoin, and vice versa for n-day rolling minimum as "n-day support".

```
# calculate the rolling maximum and minimum
data['RollingMax'] = data['Close'].shift(1).rolling(lb,
min_periods=lb).max()
data['RollingMin'] = data['Close'].shift(1).rolling(lb,
min_periods=lb).min()

# do comparisons and generate the signals (in buy/sell columns)
data.loc[data['RollingMax'] < data['Close'],
'Buy_{Lb}'.format(Lb=lb)] = 1
data.loc[data['RollingMin'] > data['Close'],
'Sell_{Lb}'.format(Lb=lb)] = -1
data['Score_{Lb}'.format(Lb=lb)] = data['Buy_{Lb}'.format(Lb=lb)] +
data['Sell_{Lb}'.format(Lb=lb)]
data = data.drop(columns=['RollingMax','RollingMin'])
```

For the weighted average of the lookback signals that we focus in our model, the score of it can be mathematically expressed as:

$$\widetilde{\Theta}_t = \vec{w} \cdot \vec{\Theta}_t = w_{i_1}\Theta_{i_1 t} + w_{i_2}\Theta_{i_2 t} + \cdots + w_{i_N}\Theta_{i_N t} \text{ such that } \sum_{j=1}^{N} w_{i_j} = 1$$

where $\widetilde{\Theta}_t$ is the weighted score at time t, N is the number of different look-backs,

$\vec{w} = (w_{i_1}\ w_{i_2}\ \ldots\ w_{i_N})^T$ are the weights of the look-back signals at time t, and

$\vec{\Theta}_t = (\Theta_{i_1 t}\ \Theta_{i_2 t}\ \ldots\ \Theta_{i_N t})^T$ are the look-back signals (-1, 0, 1) at time t.

The buy/sell signal Z_t at time t is determined by:

$$Z_t = \begin{cases} 1 & \widetilde{\Theta}_t \geq \tau_B & \textit{Buy Signal} \\ 0 & \tau_s \leq \widetilde{\Theta}_t \leq \tau_B & \textit{No Signal} \\ -1 & \widetilde{\Theta}_t \leq \tau_S & \textit{Sell Signal} \end{cases}$$

where τ_B and τ_S are the thresholds for buy and sell signals respectively

```
# calculate the weighted average of lookback signals
data['Score'] = np.zeros(len(data))
for i in range(0,len(lookbacks)):
    data['Score'] += data['Score_{Lb}'.format(Lb=lookbacks[i])] *
weights[i]
```

We further assume that there is no transaction cost, and the interest rate effect is negligible. So the investment value changes only when the investor invests all his/her capital in Bitcoin, i.e. when the latest signal is a Buy Signal. Also, the investor holds the asset until there is a Sell Signal. Mathematically, the investment value I_t $(1 < t \leq T)$ has the following property:

$$I_t = \begin{cases} I_{t-1} & \mathrm{Z}_{t-1} = -1; \text{ or } \mathrm{Z}_{t-1} = 0 \text{ and investor is not holding the asset} \\ I_{t-1} \cdot \dfrac{P_t}{P_{t-1}} & \mathrm{Z}_{t-1} = 1,; \text{ or } \mathrm{Z}_{t-1} = 0 \text{ and investor is holding the asset} \end{cases}$$

where T is the last day of the investment period,

i.e. the last day of training / testing period.

In our strategy, we will assume that the investor has 1 dollar initially ($I_1 = 1$), and we can calculate the total return in percentage.

```
# loop through all the rows (days), with a rolling investment value (init:
1)
for index, row in combined_df.iterrows():

    # change investment value if holding asset
    if holding == 1:
        investment_value *= float(row['Close'])/float(last_price)
    last_price = row['Close']

    # get next signal
```

```
    signal = row['Buy/Sell']

    # update holding status
    if signal == 1 and holding == 0:
        holding = 1
    if signal == -1 and holding == 1:
        holding = 0
```

I_T can be obtained by iterating the property of I_t through the whole training / testing period.

The total trading return Π is given by:

$$\Pi(\vec{w}, \tau_B, \tau_s) = \frac{I_T - 1}{1} \cdot 100\% = (I_T - 1) \cdot 100\%$$

Mathematically, we search for $\text{argmax}_{\vec{w},\tau_B,\tau_s} \Pi$, where Π is the trading return (in %). For simplicity, we use a grid search to find the best parameters, i.e. we iterate through all the possible values of the parameters and choose the onces with the highest trading return. The lookbacks determining $\vec{\Theta}_t$ are predetermined as 20, 30, 60 and 90 days in our model for simplicity (to avoid super-long runtime in training), i.e. we adopt $N = 4$, and $(i_1, i_2, i_3) = (20,30,60,90)$. For parameters $\vec{w} = (w_{i_1}\ w_{i_2}\ \ldots\ w_{i_N})^T$ and $\{\tau_B, \tau_s\} \in [-1,1]$, we assume:

$$\vec{w} \in \{0.1, 0.2, \ldots, 1.0\}^4 \in \mathbb{R}^4 \text{ and}$$

$$(\tau_B, \tau_S) \in \{0.3, 0.5, 0.7, 0.9\} \times \{-0.3, -0.5, -0.7, -0.9\}$$

to yield the greatest trading return on the backtest of training data using grid search.

```
# Generate Parameter Lists for Grid Search
# a list of all possible weights (bundle)
weights_list = []
def sums(length, total_sum):
```

```
    if length == 1:
        yield (total_sum,)
    else:
        for value in range(total_sum + 1):
            for permutation in sums(length - 1, total_sum - value):
                yield (value,) + permutation
ww = list(sums(len(lookbacks),10))
for w in ww:
    weights_list.append(tuple(w1/10 for w1 in w))

# a list of all possible thresholds (bundle)
thres_list_1 = [[0.3,0.5,0.7,0.9],[-0.3,-0.5,-0.7,-0.9]]
thrs_list = list(itertools.product(*thres_list_1))
```

```
# Grid Search for best weights and thresholds
for weights in weights_list:
    percentage = round(count*100/len(weights_list),2)
    print('Progress: {Percentage}%'.format(Percentage=percentage),
end='\r')
    count += 1
    for thrs in thrs_list:
        rt = get_return_lb(data_signal,
currency,lookbacks,weights,thrs,'train')
        if rt > max_rt:
            max_rt = rt
            max_weights = weights
            max_thrs = thrs
```

This model adopts the following training and testing periods:

- Training Period (3 years): 2017-01-01 to 2019-12-31
- Testing Period (1 year): 2020-01-01 to 2020-12-31

```
lookbacks = 20,30,60,90 # days
# define the training and testing period
train_period = dt.datetime(2017,1,1).date(),
dt.datetime(2019,12,31).date()
test_period = dt.datetime(2020,1,1).date(),
dt.datetime(2020,12,31).date()
```

Regarding the result of the grid search, the training result indicates:

- 1145.01% return in 3 years with $\vec{w} = (0.0,0.0,0.3,0.7)^T$ and $(\tau_B, \tau_S) = (0.5, -0.3)$, which is equivalent to 131.77% return p.a.

```
From 2017-01-01 to 2019-12-31
Training Return: 1145.01% with parameters w:(0.0, 0.0, 0.3, 0.7) and t:(0.5, -0.3)
```

The testing result using $\vec{w} = (0.0,0.0,0.3,0.7)^T$ and $(\tau_B, \tau_S) = (0.5, -0.3)$, indicates:

- 137.27% return in one year, which is not far from the training return 131.77% (p.a.).

```
From 2020-01-01 to 2020-12-31
Testing Return: 137.27%
```

We conclude that the over-fitting in this model is not severe, and the strategy is working well and producing a considerable return.

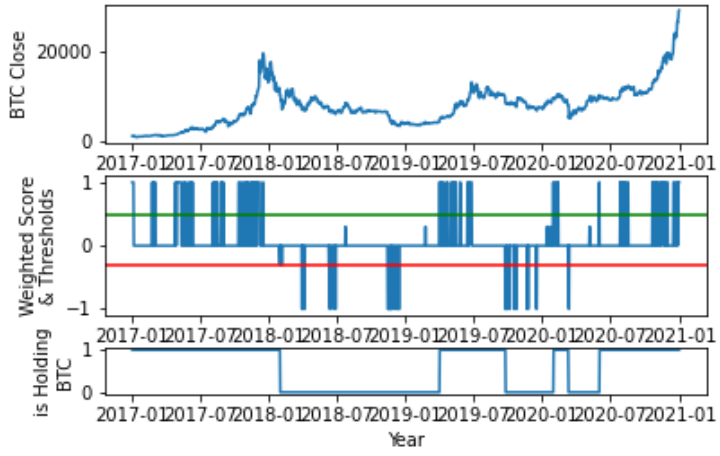


Figure Results: Training & Testing Periods

We also found that longer lookback signals tend to be more insightful, as we see the weight on the longest signal is the highest after grid search. Trading with the weights and thresholds chosen by the grid search in the current result is indeed equivalent to "buy when **60-day** signal says buy; and sell when **90-day** signals says sell". The grid search can be further improved in the future to include more possible values, but the runtime to train the model will be much longer in that case. For the signals in our model, we do not use even longer signals, as (i) the training and testing periods are short, (ii) Bitcoin has only been available for a few years (so we cannot have more data), and (iii) the general behavior of Bitcoin we want to examine may only be happening after 2017. Before 2017, Bitcoin has a comparatively very low price, possibly because of the lack of awareness and acceptance. We try to extend our model to earlier periods (prior to 2017) and result is as follows:

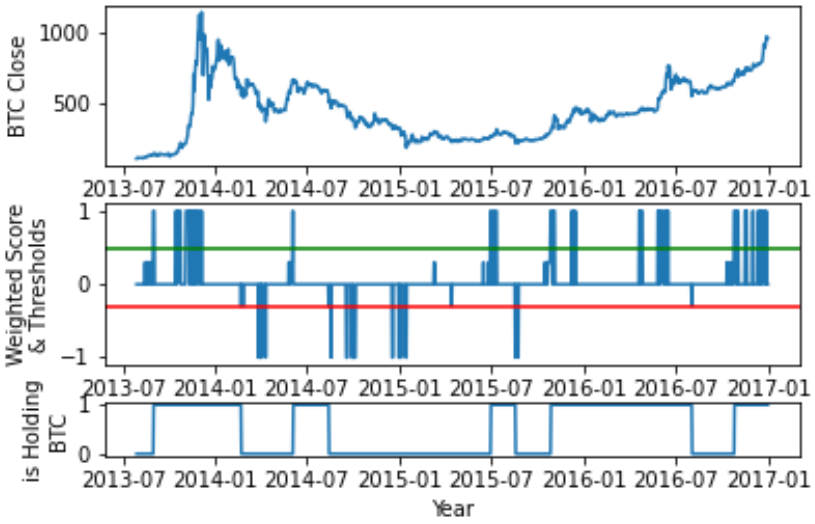


Figure Results: Before 2017 (before the training & testing periods)

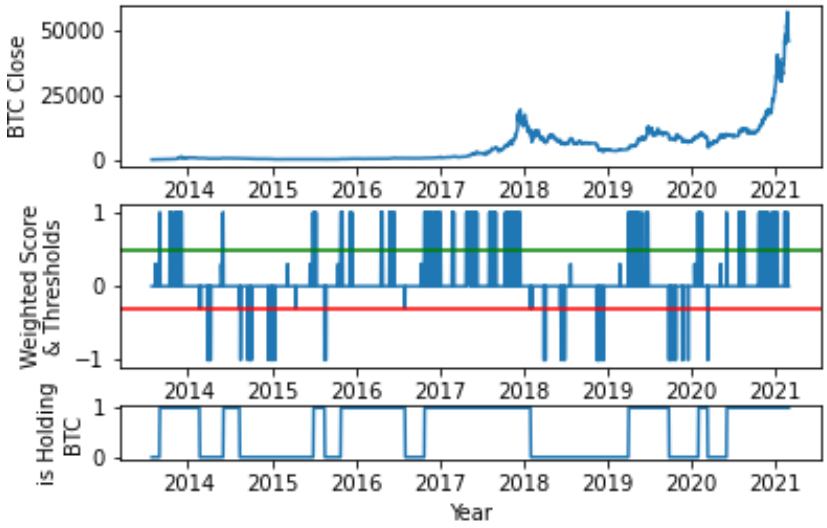


Figure Results: All Closes staring from 2013

AppendixV: Detailed Explanation of Sentiment Analysis using LSTM

For full codes of this model (Part III), please refer to these files:

part_3a_sentiment_data_preprocessing.ipynb

part_3b_sentiment_analysis return_calculation.py

We have researched for different methods including VADER and TextBlob for evaluating the sentiment scores, and finally we chose VADER to be included in our model because it is attuned specifically to sentiments expressed in social media (e.g. Twitter) and is able to give more detailed information (e.g., 'positive', 'negative', 'compound', 'neutral'). VADER uses a list of lexical features (e.g. word) which are labeled as positive or negative according to their semantic orientation to calculate the text sentiment and returns the probability of a given input sentence to be positive, negative, and neutral. After we obtain the sentiment score of each

tweet, we then calculate the daily average sentiment score by taking the mean of all the tweets on each day using the following codes:

```
import pandas as pd
import numpy as np
import nltkfrom nltk.sentiment.vader import
SentimentIntensityAnalyzer
from tqdm import tqdm
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()
text_data_score = text_data['text'].progress_apply(lambda Description:
sid.polarity_scores(Description))
score = text_data_score.progress_apply(lambda
x:pd.Series(ast.literal_eval(x)))
binned_score = score.resample('D').mean()
```

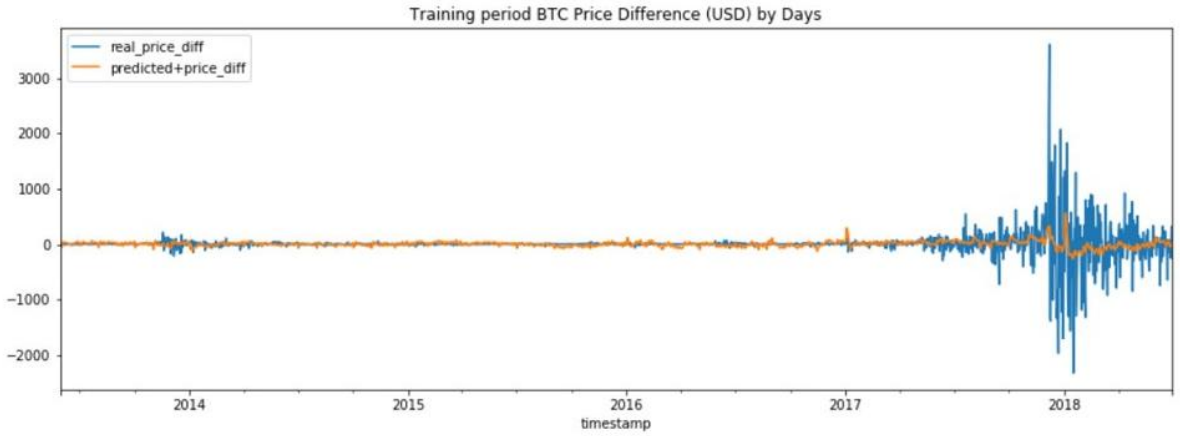
We then train the Long short-term memory (LSTM) model with the daily sentiment score (calculated in the previous step) and price change history (from the dataset) in the past 30 days to predict the price change in the next day. LSTM is an artificial recurrent neural network (RNN) architecture with feedback connections, which is especially good at making predictions based on time series data even with varying lags between input samples. It could also deal with the vanishing gradient problem that could happen when training traditional RNNs. The data input is normalized to minimize the impact of extreme values on the model. Our model uses the mean square error as a loss function and Adam as the optimization method. We performed grid-search with cross-validation to fine-tune the best parameter, which we found is 256 batch size and 200 epochs.

```
def buildManyToOneModelFn(shape):
    def buildManyToOneModel():    model =
Sequential()    model.add(LSTM(10, input_length=shape[1],
input_dim=shape[2]))    model.add(Dense(1))    model.compile(loss="m
se", optimizer="adam")
    model.summary()
    return model
return buildManyToOneModel
model = KerasRegressor(build_fn=build_fn, verbose=0)
batch_size = [ 64, 128 ,256]
epochs = [100, 200, 500]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1,
cv=10)
grid_result = grid.fit(X_train, Y_train)
```

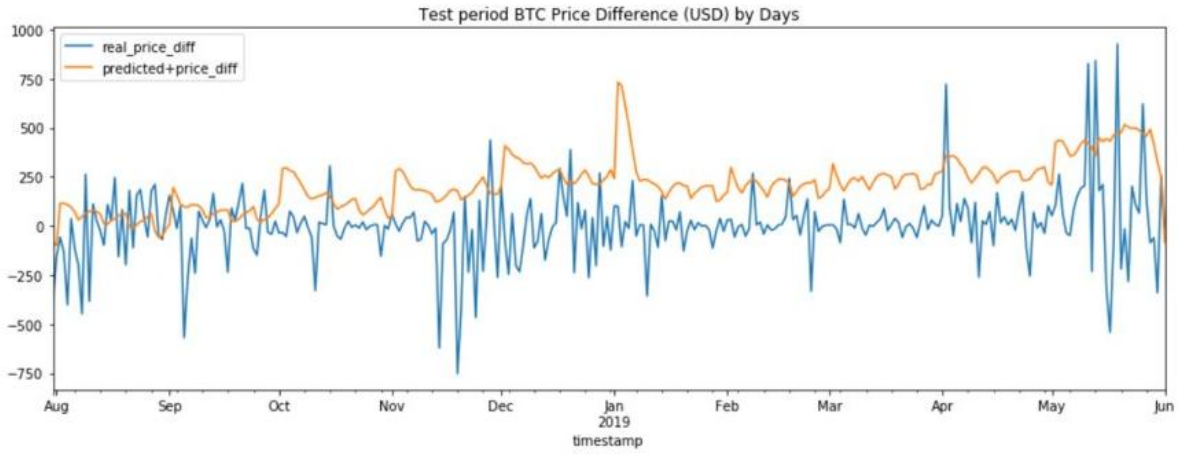
Finally, we normalize the predicted price change into the range between -1 and 1. If the score is closer to 1 on a particular day, it indicates that buying the asset (Bitcoin) has a higher probability of getting a higher positive return.

```
trade_score = np.reshape(np.array(result['predicted+price_diff']),
(len(result),1))
scaler = MinMaxScaler(feature_range=(-1, 1))
trade_score = scaler.fit_transform(trade_score)
trade_score = pd.DataFrame(trade_score,columns =
['Score'],index=result.index)
```

Visualization of the Training Period



Visualization of the Testing Period



As we observe from the figures above, we observe that the prediction in price difference is not very accurate and tend to be a bit higher than actual. Thus, the model has to be further tuned in the future, or that tweets is not a determining factor of the price of Bitcoin.

The result of this model is as follows:

```
get_return(trade_score[['Score']],'Bitcoin',pd.to_datetime('2018/8/1',utc=True),\
           pd.to_datetime('2019/6/1',utc=True),0.7,-0.7)
```

```
117.17290142238267
```

using the threshold as (0.7,-0.7). The return is comparatively low to other models.

We will see if the sentiment factor is useful when attempting to combine different factors in the combined model in the next part.

Appendix VI: Detailed Explanation of the Combined Model

For full codes of this model (Part IV), please refer to these files:

part_4_scores_combined.ipynb *return_calculation.py*

We attempt to linearly combine (sum up with a weight) the scores we obtain from the three aforementioned models. The weighted score $\vec{s}_w$ can be mathematically expressed as

$$\vec{s}_w = a\vec{s}_1 + b\vec{s}_2 + c\vec{s}_w \ \ where \ \ a + b + c = 1$$

We look for a, b, c which the weighted scores can produce the greatest backtest return using a pair of threshold, and this pair of threshold is done by grid search. For simplicity, we assume that a, b, c and the thresholds are:

$$a \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

$$b \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

$$c \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$$

$$(\tau_B, \tau_S) \in \{0.3, 0.5, 0.7, 0.9\} \times \{-0.3, -0.5, -0.7, -0.9\}$$

$$where \ a + b + c = 1$$

The result of the grid search of best parameters a, b, c and thresholds are as follows:

```
From 2018-07-31 to 2019-01-31
The total return is 1071.28% with parameters a=0.2, b=0.2, c=0.6 and threshold=(0.3, -0.3)
```

We observe that this combined method produces a comparatively high return, and model three (sentiment) has the highest weight. We further deduce that a combination of results from different model may increase the total insightfulness of the trading signals and provide a

better indication for the low frequency trading of Bitcoin. The resulting return of 1071.28% is equivalent to ~12736% p.a. is thus better than using different models individually.

Date	Score1	Score2	Score3
2018-07-31	0.361471	0.0	-0.964151
2018-08-01	0.506797	0.0	-1.000000
2018-08-02	0.515446	0.0	-0.484861
2018-08-03	0.306170	0.0	-0.479721
2018-08-04	0.436978	0.0	-0.502476

Combining the Three Columns

References

Bitcoin Price Dataset:

SRK. 2021. Cryptocurrency Historical Prices Dataset. *Kaggle*, 27 February 2021. [Online]. Available: https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory

Bitcoin Tweets Dataset:

Alex. 2020. Bitcoin tweets - 16M tweets. *Kaggle*. [Online] Available: https://www.kaggle.com/alaix14/bitcoin-tweets-20160101-to-20190329

Cryptocurrency Market Figures Referenced:

Y. Barabash. 2021. Cryptocurrency Investments: New Times, New Opportunities. Finextra, 22 February 2021. [Online]. Available: https://www.finextra.com/blogposting/19926/cryptocurrency-investments-new-times-new-opportunities