CQF 2022 - FINAL PROJECT

Deep Learning Assignment - LSTM to predict Bitcoin up/down movements

ENZO YAMAMURA

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Long Short Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) algorithm capable of learning both long and short dependencies over time. Therefore, it is known to perform well when time dependency is observed. This study aims to predict up/downward moves in Bitcoin returns by using a multivariate LSTM classificator with Reddit sentiment and technical indicators as features.

1 INTRODUCTION

FAMA et al's [1, 2] Efficient Markets Hypothesis establishes that the asset prices reflect all available information at any given time. Therefore, all price oscillations must come from novel information, which, by definition, are impossible to predict. Hence, all asset prices should follow a random walk, with the best future price estimator being its current value, meaning an accuracy over 50% would be impossible.

According to Kahneman and Tversky [3], financial strategies are not, in fact, decided upon fundamentals and logic alone, but also upon perceived risk and emotions. Nowadays, both information and emotions became widespread due to social media and online news portals. In numbers:

Twitter numbers (2021) [4]:

- 199 million users
- 500 million tweets per day
- One every 5 american adults used the platform
- 280 letters limitation

Reddit numbers (2021) [5]:

- Over 52 million users active daily
- One every 5 american adults used the platform
- Each commentary has a 40.000 character limitation (more information)
- 2 billion comments registered in 2020 (after discarding spam)

Therefore there is an increasing amount of available data when it comes to either news or sentiment. Hedge funds already monitor social media [6] as proxy for market sentiment. Furthermore, Elon Musk's tweets effect on cryptocurrency prices [7] and the the Gamestop Short Squeeze [8] reiterate the impact of social media on assets. Besides, news tend to show up faster online, such as the US Airways plane crash in the Hudson river in 2009 [9], which was first reported on Twitter instead of television.

Previous research, such as Nguyen and Shirai [10], Abraham et al [11] and Mohaptra et al [12], used social media and news as proxies for market sentiment to predict asset movements, and have reached accuracy metrics over 50%, which means returns above the EMH's 50% may be possible when using these new sources of data.

This work takes inspiration from the above and the CQF lessons, and aims to predict Bitcoins' daily up/down movements by using technical indicators and Reddit sentiment as features and a Long Short Term Memory Deep Learning architecture. Additionally, a simple LSTM regressor is built with sentiment alone as a feature. The structure is as follows:

- Methods
- Databases Processing, Feature Engineering, Data Analysis and Feature Selection
- Model Training and Results
- Conclusion

All the code is available in the GitHub repository: `https://github.com/Yamamuen/CQF_final_project` with instructions on how to run the code and plenty of comments. Please e-mail `enzo.yamamura7@gmail` after evaluation so the repository can be set to private. Let me know if you have any issues to access it when evaluating the submission.

2 METHODS

In the section all the technical information used is explained, from Deep Learning to Python libraries used.

2.1 Deep Learning

Neural Networks are bio-inspired algorithms based on how neurons function: dendrites receive information as synapses and axons process and and propagate it to other connected neurons. The basic structured is demonstrated below: input data is provided, which is then processed in the hidden layers and the results are displayed in the output layer. Each connection (line) has a weight, and at every iteration the information is propagated until the output layer, where an activation function transforms the received data into an output.

Figura 1 – Redes Neurais

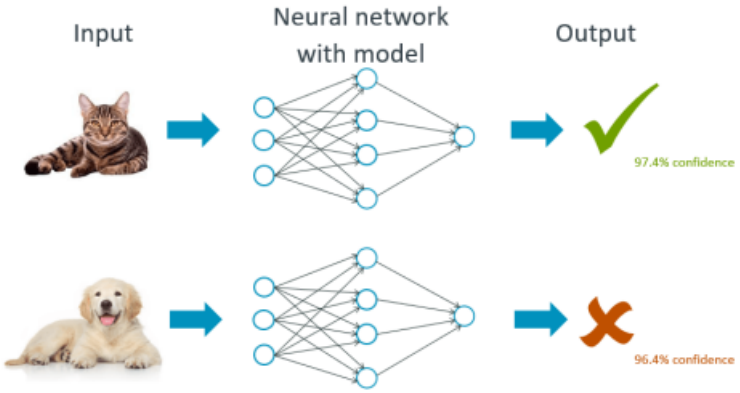

Figure 1: Simple Neural Network Structure. Source: Medium Article.

In Neural Networks the neurons are independent and there is no memory (thus inputs and outputs are independent of one another), disallowing their usage for when the sequence matters. Rumelhart, Hinton and Williams (1996) [13] created the Recurring Neural Networks (RNN) model; RNN's name is self-explanatory and it circumvents vanilla Neural Networks limitations by adding *backpropagation*: at each iteration the error is computed to adjust the connection weights retroactively, re-starting the process and minimizing the loss function (through gradient descent) until the input data can generate the outputs reasonably well.

Figura 2 – RNN x ANNs.

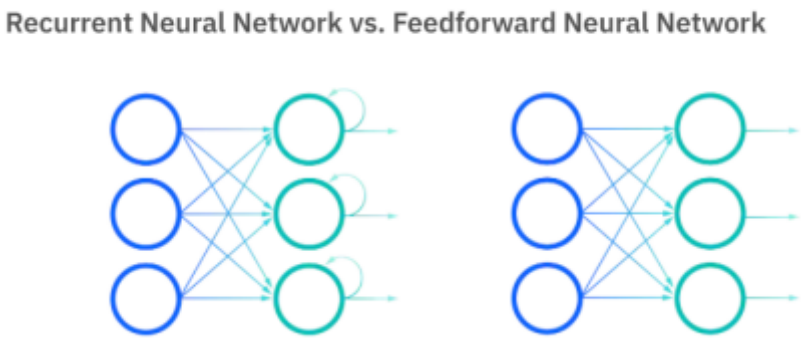

Figure 2: RNN. Source: IBM.

However, RNNs have a shortcoming: they have limited memory, failing when the problem involves carrying information over long series. Hochreiter and Shmidhuber (1997) [14] developed the Long Short Term Memory algorithm, a subclass of RNN capable of learning dependencies over long series. It comprises *forget gates* which are trained according to informational gain, filtering out which previous information should be discarted or propagated to the next layers. The figure below illustrates the idea.

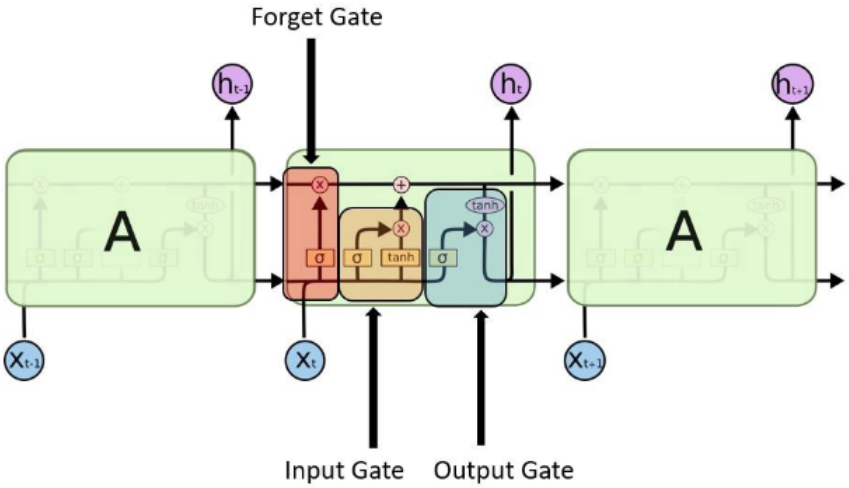

Figure 3: LSTM Structure. Source: Medium Article.

This study assumes Bitcoin has a time dependent relationship, therefore the Long Short Term Memory algorithm is used.

2.2 Sentiment Analysis

Text data available in social media is non-structured. Meaning it is context-specific and made to be consumed by humans. The Natural Language Processing (NLP) field aims to yield information from such data, with many different methods to make unstructured data consumable by computers [11]. In NLP there are Sentiment Analysis tools, aimed at extracting emotions from text with intricate data processing and machine learning algorithms. In a broad sense Sentiment Analysis can be divided into machine learning based approaches and *lexicon* based methods (dictionaries).

VADER (Valence Aware Dictionary and sentiment Reasoner) is one of the latter [15]: it is a lexicon specially calibrated to social media, boasting of good performance on Sentiment Analysis over Twitter, Facebook and Reddit data. It detects polarity (positive, neutral, negative) for each word and is also capable of interpreting internet-specific characters, such as emojis, symbols, contractions, slangs and CSS formatting. It alleviates the toil of processing text data too much before yielding sentiments. It was compared to other 11 sentiment analysis tools (2015) [15] and had the best performance. To analyse Reddit sentiment, this project uses VADER.

2.3 Technical Indicators

Besides sentiment, technical indicators were calculated in order to be used as features. The formulae are available in plenty of websites, but the used herein (all of which were implemented with TA-lib) are listed bellow. Lookback periods of 5, 10, 21 and 50 were calculated for each, but the other parameters were set to their default values.

Exponential Moving Average:

$$EMA_t(p) = \alpha \cdot p_t + (1 - \alpha) \cdot EMA_{t-1},$$

where α is the exponential smoothing, set to default value of 2 and the lookback calculated for [5, 10, 21 50] D. It yields a trend, as the Moving Average, but placing greater height on more recent observations.

Average True Range:

$$ATR_t = \frac{ATR_{t-1} \times (n-1) + TR_t}{n}$$

$$TR = \max(high, close_{t\text{-}1}) - \min(low, close_{t\text{-}1})$$

which is basically a volatility measure.

Stochastic Oscillator:

$$\%K = 100 * \frac{(close_t - low_N)}{(high_N - low_N)}$$

is a momentum indicator based on resistance and support.

Moving Average Convergence Divergence:

$$MACD = EMA(M) - EMA(N))$$

is the difference between EMA over M days and EMA over N days. Here M and N are set to their usual values, 12 and 26, respectively.

Commodity Channel Index:

$$TP_t = p_t = \frac{high_t + low_t + close_t}{3},$$

$$CCI_t = \frac{1}{0.015} \cdot \frac{p_t - SMA_n(p_t)}{MAD_n(p_t)}$$

with SMA being the Simple moving average. It was designed to detect beginning and ending market trends.

Accumulation/Distribution Indicator:

$$A/D_t = A/D_{t-1} + MFM_t, \text{where}$$

$$MFMt = \frac{Close - Low) - (High - Close)}{High - Low}$$

which gauges supply/demand to assess whether the stock is being accumulated or distributed.

Bollinger Bands:

$$\text{Upper Bollinger Band} = SMA(close, N) + \text{number of standard deviations} * st.dev(close, N)$$

$$\text{Lower Bollinger Band} = SMA(close, N) - \text{number of standard deviations} * st.dev(close, N)$$

yields price dynamics by gauging upper and lower bands using moving averages and standard deviations. N is changed for the previously mentioned lookbacks but the other parameters are kept at their default values (number of std deviations =2).

Momentum Indicator:

$$MOM = Price_t - Price_{t-N}$$

yields momentum, or simply the price difference between time t and N lookbacks.

Relative Strength Index:

$$RSI = 100 * \frac{EMA_N(p) \text{ of } U}{EMA_N(p) \text{ of } U + EMA_N(p) \text{ of } D}$$

U/D being, respectively, upward and downward changes over the period. If prices do not change, both are set to 0. It is a momentum indicator that captures both magnitude and velocity of price movements.

Drift-Independent Volatility Estimator:
Created by Yang and Zhang [16], it is considered one of the best measures for stock price volatility, with the added feature of considering overnight jumps. As this formula was not available in the TA-lib package, the following code was adapted from this link:

```
def get_hvol_yz(data, lookback=10):
    o = data.Open
    h = data.High
    l = data.Low
    c = data.Close

    k = 0.34 / (1.34 + (lookback+1)/(lookback-1))
    cc = np.log(c/c.shift(1))
    ho = np.log(h/o)
    lo = np.log(l/o)
    co = np.log(c/o)
    oc = np.log(o/c.shift(1))
    oc_sq = oc**2
    cc_sq = cc**2
    rs = ho*(ho-co)+lo*(lo-co)
    close_vol = cc_sq.rolling(lookback).sum() * (1.0 / (lookback - 1.0))
    open_vol = oc_sq.rolling(lookback).sum() * (1.0 / (lookback - 1.0))
    window_rs = rs.rolling(lookback).sum() * (1.0 / (lookback - 1.0))
    result = (open_vol + k * close_vol + (1-k) * window_rs).apply(np.sqrt) * np.sqrt(252)
    result[:lookback-1] = np.nan

    return result * 100
```

Apart from the drif-independent volatility, all the indicators were readily available in the TA-lib library.

2.4 Python Libraries

Lib	Usage
TA-lib	Provides the main technical indicators easily
Fast Text	NLP to identify language
VADER	Sentiment Analysis toolkit
Tensorflow	Besides being a framework for building DL, with CUDAS, CUDNN and Keras allow DL algorithms to run on GPU
Talos	Simple and effective hyperparameter tuning tool
Boruta	Feature selection library that builds upon other FS methods (such as DTR), enhancing the selection by generating shadow copies of the data and testing for information gain

Table 1: Libs used

3 DATABASES PROCESSING, FEATURE ENGINEERING, DATA ANALYSIS AND FEATURE SELECTION

3.1 Datasets

Two datasets were used: a Reddit Comments BigQuery Dataset and Bitcoin OHLC daily dataset. The first has 1.7 billion comments from 2015 to 2019 and the latter has daily OHLC Bitcoin in USD from 2014 to 2021. No adjustments were required in the latter.

In order to retrieve only comments related to Bitcoin, the following query was used on the BigQuery database:

```
SELECT
  subreddit,
  created_utc,
  body,
  score
FROM
  [reddit-btc-analysis:comments.reddit_btc_comments]
WHERE
  (LOWER(body) LIKE '% bitcoin%'
    OR LOWER(body) LIKE '% bitcoin %'
    OR LOWER(body) LIKE '% btc.%'
    OR LOWER(body) LIKE '% btc %')
```

Figure 4: Query used.

Which yielded a database with over 5 million comments mentioning Bitcoin.

3.2 Data Processing and Feature Engineering

All the processing done to retrieve Reddit sentiments is in this notebook. The following steps were performed (took over 12 hours with parallel processing):

- Special characters and formatting were removed
- Language was identified with Fast Track in order to keep only English text (VADER limitation)
- VADER's compound polarity was calculated for each comment
- The comment score (up - downvotes) was used to weight each polarity in order to gauge its influence

The technical indicators were all computed with TA-lib for all the looback periods of 5, 10, 21 and 50 days, and the target variable (up/down) return movements was created. For more details on the code and steps refer to: `https://github.com/Yamamuen/CQF_final_project/blob/main/processing_EDA.ipynb`. Drift-independent volatility was calculated with the function defined in the Method section.

In order to use both, the final Reddit dataset was aggregated and merged on days to the Bitcoin time series. During the aggregation, the polarities' mean, median, sum and volume of comments were calculated.

3.3 Data Analysis

A thorough Data Analysis was performed to check for trends and relations between variables.

Bitcoin's price behaved as follows during the considered time span:

Figure 5: BTC over time.

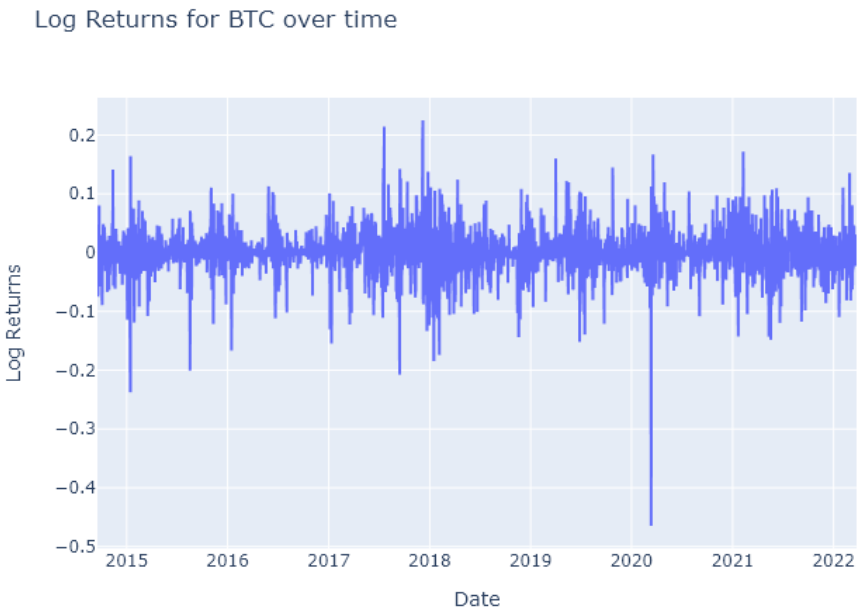

Figure 6: BTC Log Returns over time.

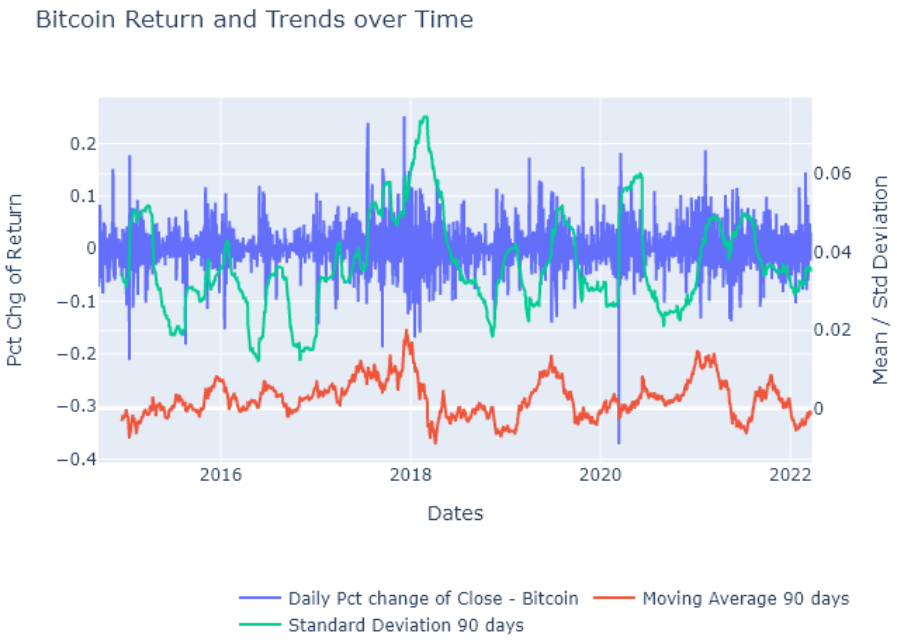

Figure 7: BTC Stationarity over time.

It is clear that its series is far from stationary, with both moving average and standard deviation being far from constant, which instantly discards the usual ARIMA time series algorithms as viable alternatives. Then, the Reddit Sentiment data was analysed:

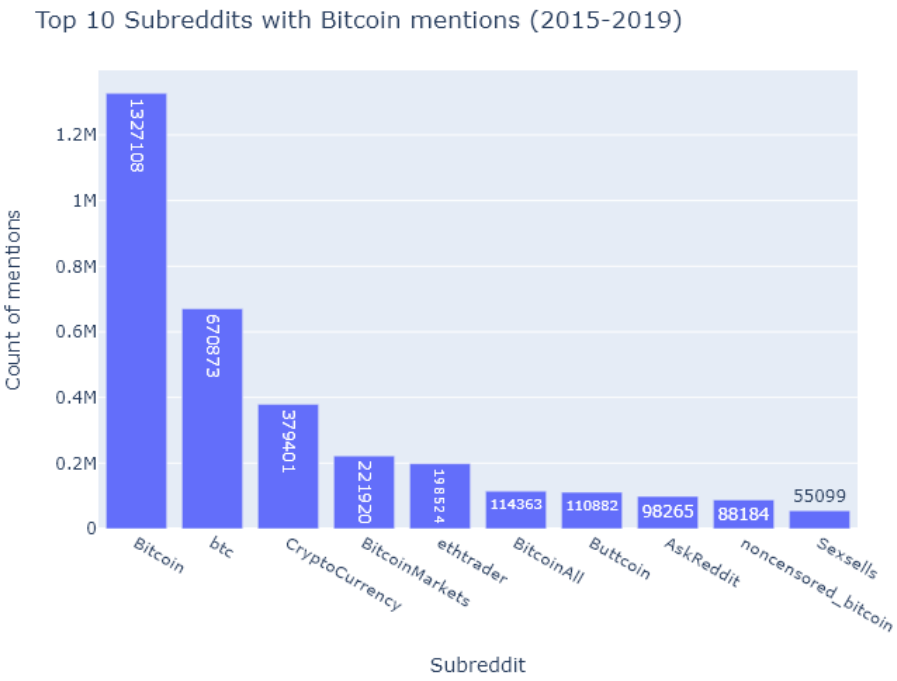

Figure 8: Subreddits with the most BTC mentions.

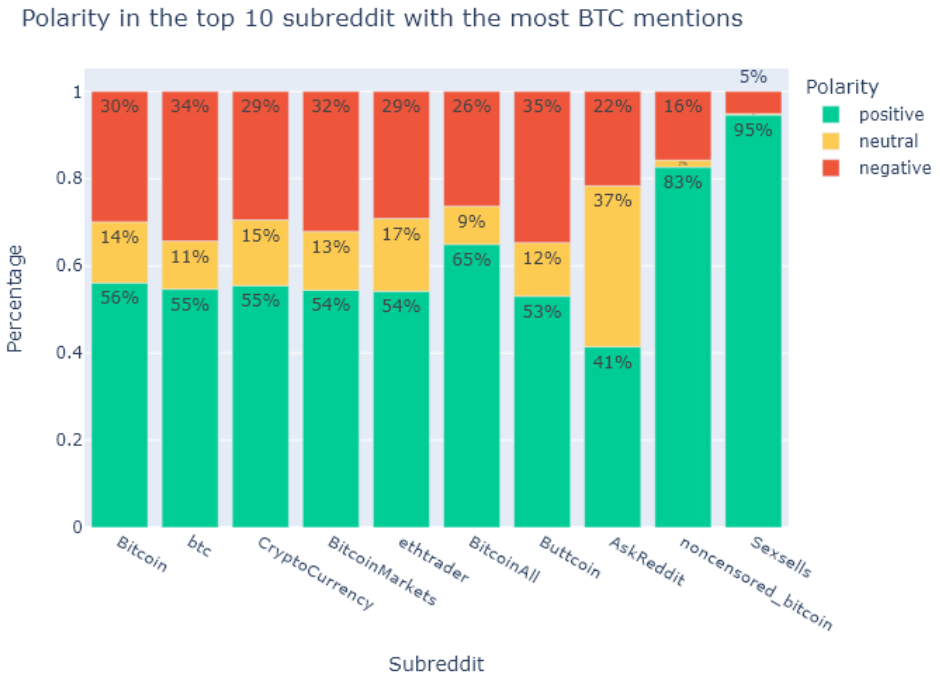

Figure 9: Polarity distribution per Subreddit.

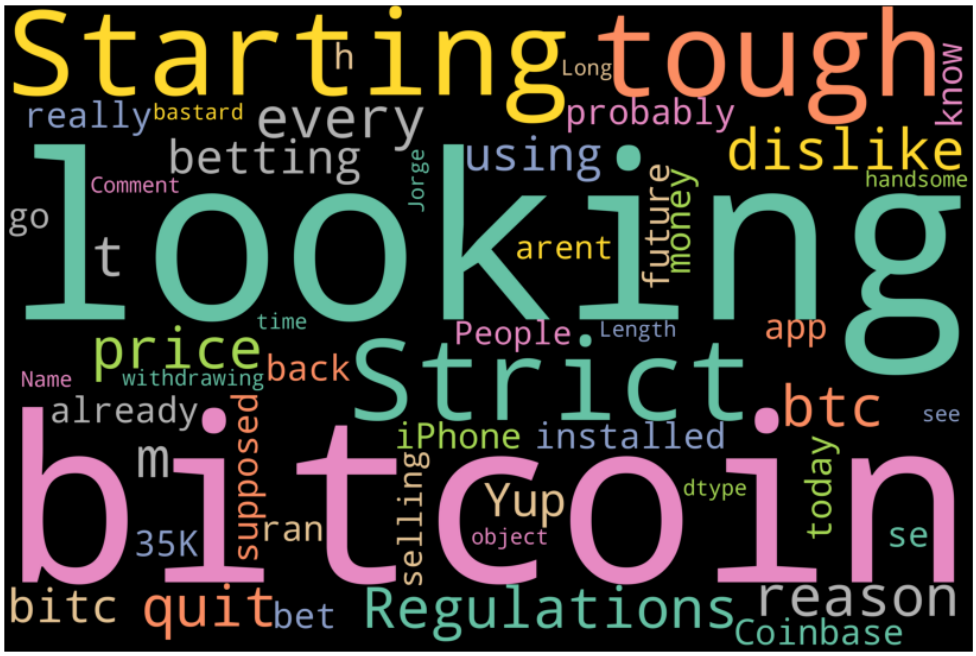

Figure 10: Words most frequently associated with BTC mentions on Reddit.

The wordcloud shows words related to people seeking help ("Starting", "looking") and words related to regulations and opinions, proving it could be an effective proxy for market sentiment. Furthermore, some of the top 10 subreddits with the most mentions to Bitcoin seem to be positively biased, such as the last two (sex workers forums), which may be because some of those attract people that tend to see it as a value reserve instead of a speculative asset to profit from. Aggregating sentiment polarities by day, the following statistics are obtained:

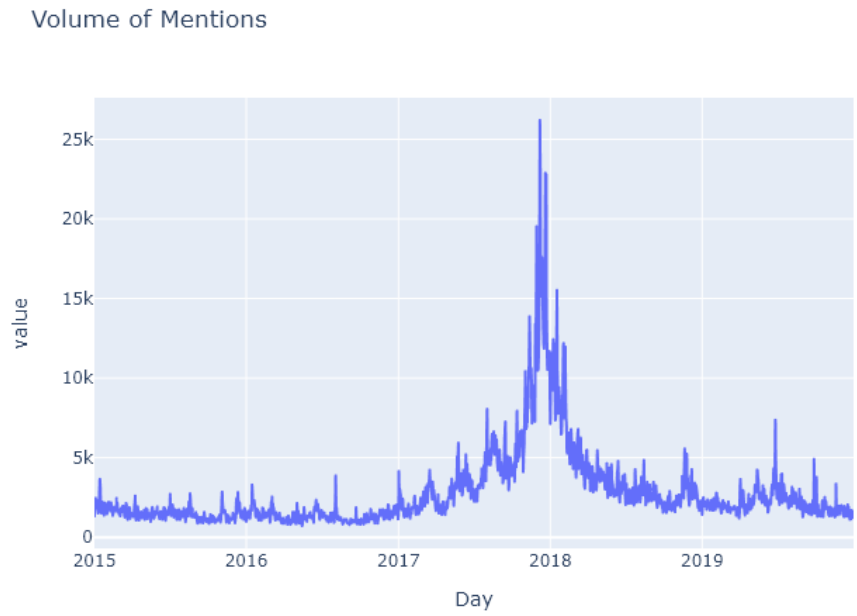

Figure 11: Volume of BTC mentions per day.

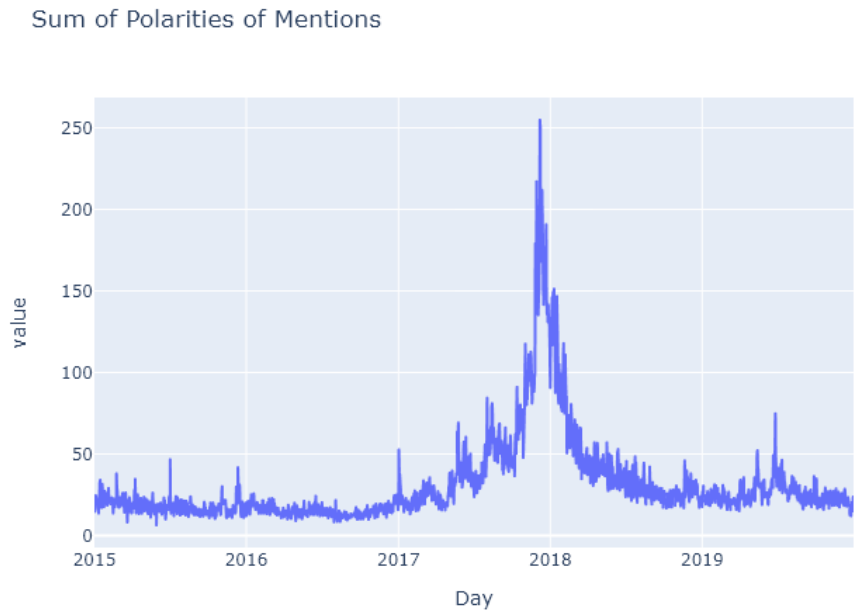

Figure 12: Aggregated Sum of Polarities of mentions per day.

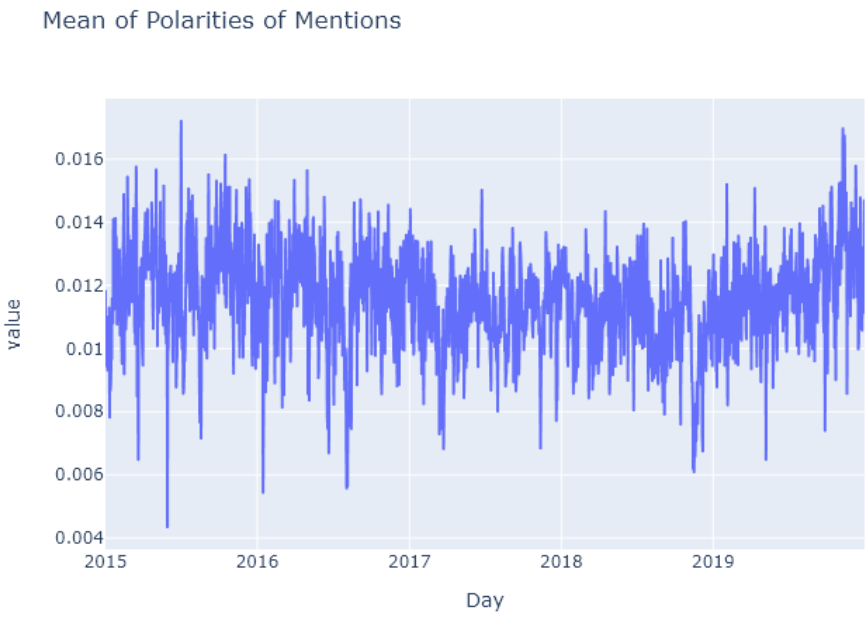

Figure 13: Aggregated Mean of Polarities of mentions per day.

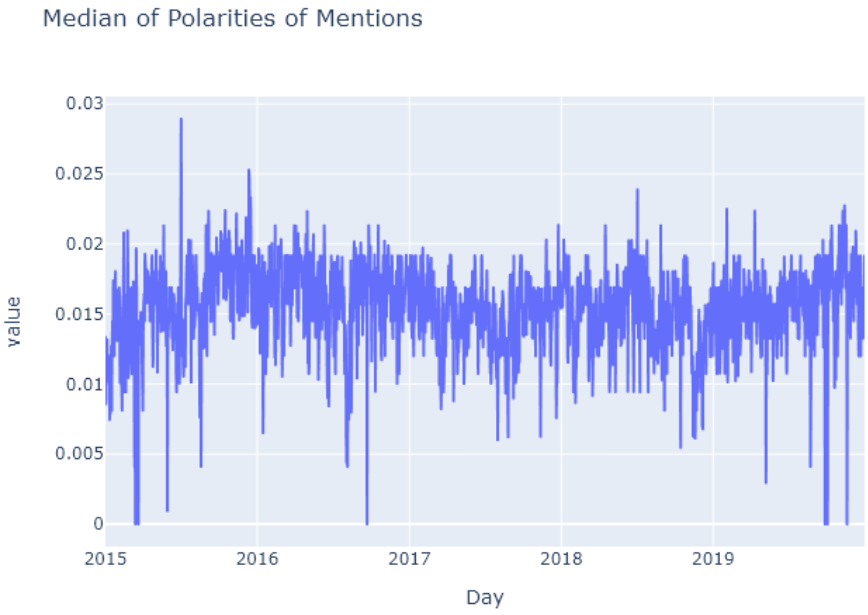

Figure 14: Aggregated Median of Polarities of mentions per day.

The mean daily polarity (already weighted by influence, check Feature Engineering section) shows a lot of volatility, but the aggregated sum o weighted sentiments seems to follow the volume of mentions evolution. Further analysis is done to check how both sentiment and Bitcoin moved over time:

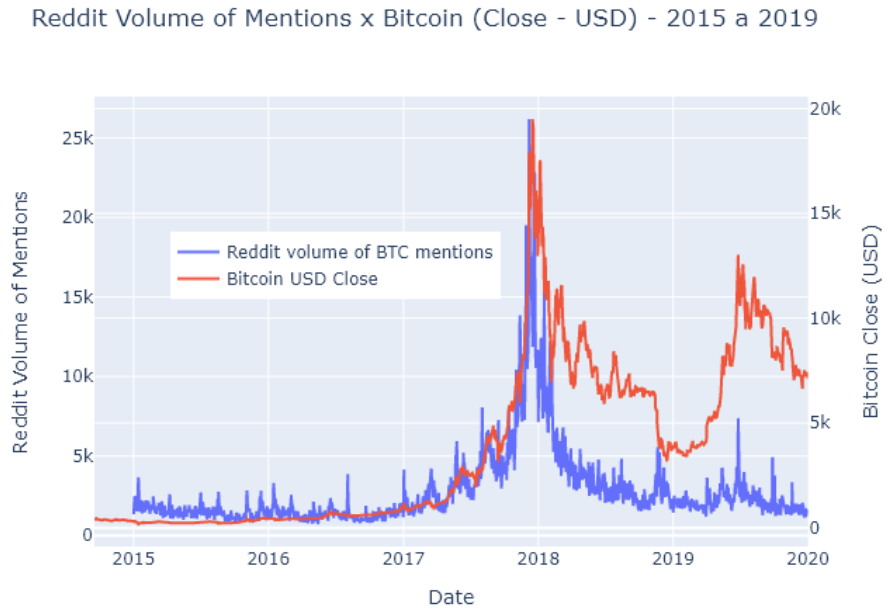

Figure 15: Bitcoin Close x Volume of Reddit Mentions.

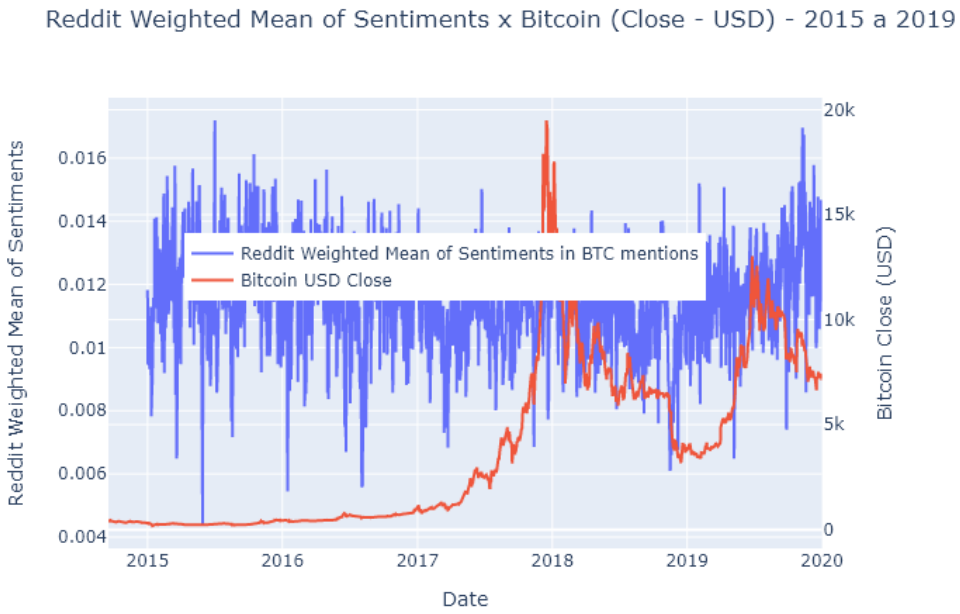

Figure 16: Bitcoin Close x Mean of Polarity for Reddit Mentions.

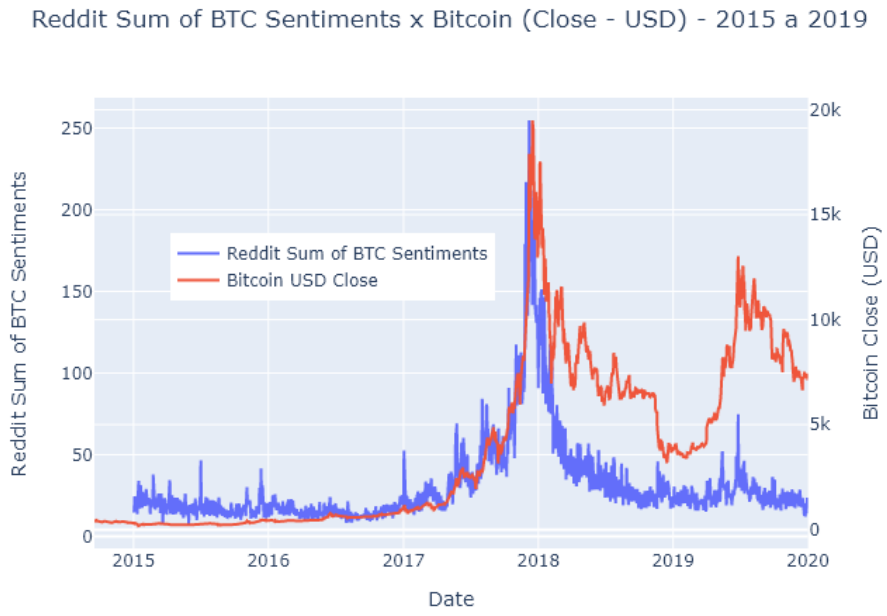

Figure 17: Bitcoin Close x Aggregated Sum of Polarity for Reddit Mentions.

It is clear, specially considering Reddit Sentiment daily sums and volume, that both time series seem to follow each others trends. Regarding other variables, from all the technical indicators mentioned and all the different lookbacks (5, 10, 20 and 50 days) way too many features were created, making it difficult to spot individual correlations in the matrix:

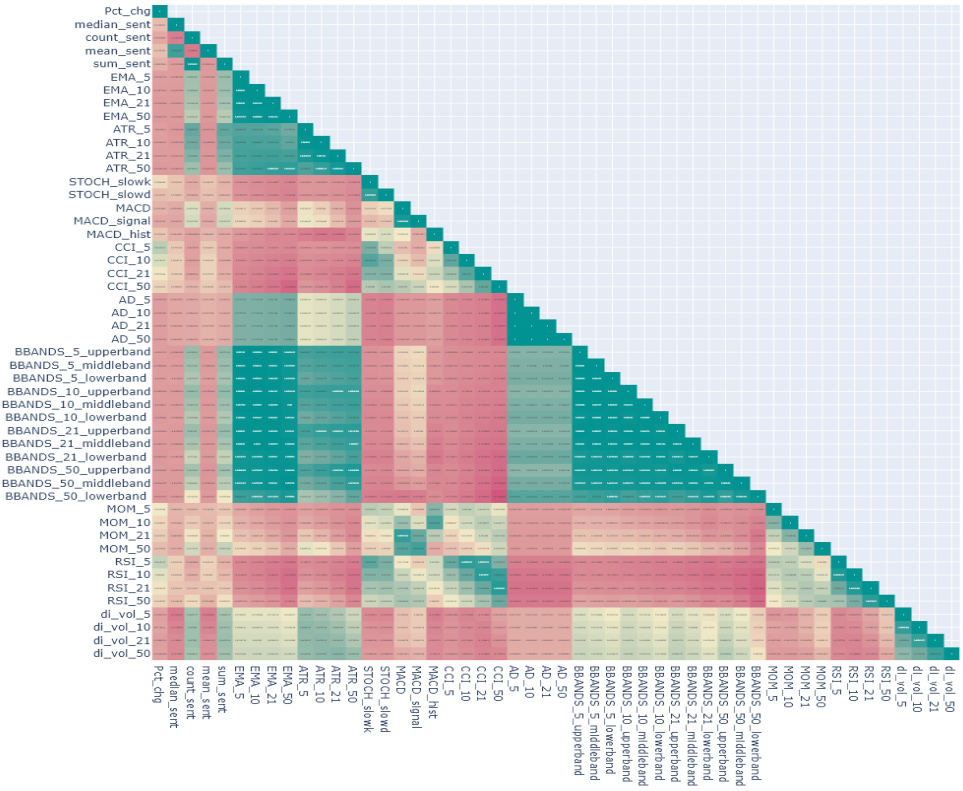

Figure 18: Correlation Matrix - All variables.

As the target variable is categorical (up/down return movements), in order to grasp correlation, the percentage return change is considered. Checking Reddit Sentiment features' correlation to returns separately:

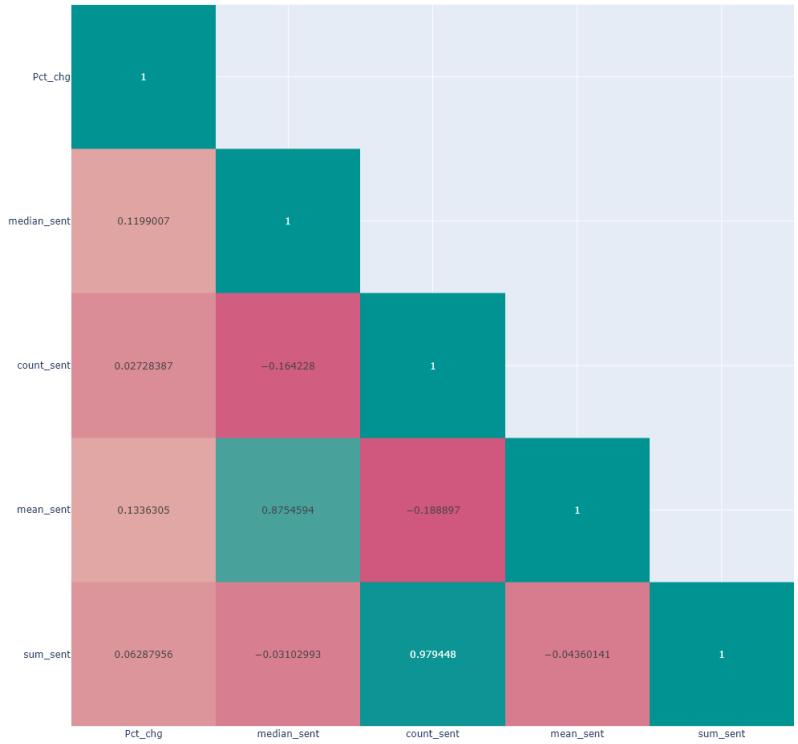

Figure 19: Correlation Matrix - Reddit Features.

Although daily volume and sum of mentions seemed to accompany BTC-USD oscillations over time in the previous visualizations, here both median and mean (redundant) of polarities boast the higher correlations to BTC. The top 10 most correlated to BTC returns are momentum indicators, such as MOM, RSI and CCI:

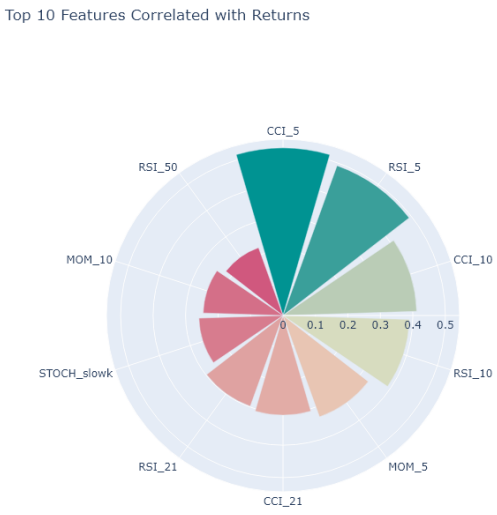

Figure 20: Top 10 features most correlated to BTC returns.

3.4 Feature Selection with Boruta

Boruta is a tool for automated feature selection. In a nutshell, the way it was used here, it supercharged the Random Forest Classifier's feature importance method by adding a *shadow* (or copy) dataset of features generated by random shuffling each; the valid features are selected among the shadow features that carry the most information.It takes an all relevant feature selection approach, which means "...it tries to find all features carrying information usable for prediction, rather than finding a possibly compact subset of features on which some classifier has a minimal error" (from its GitHub). Alternative methods (such as SOMs) exist for feature selection and/or dimension reduction, but Boruta was chosen both due to its efficiency and explainability. Besides, it is pretty straightforward to implement. The code excerpt bellow shows how it was implemented.

```
#Defining class weights (to deal with class imbalance):
def cwts(data):
    c0, c1 = np.bincount(data['target'])
    #making the weights inversely proportional to the amount of observations:
    w0, w1 = (1/c0)*(len(data))/2, (1/c1)*(len(data))/2
    return {0: w0, 1:w1}

# Define random forest classifier with the weighted classes:
forest = RandomForestClassifier(n_jobs=-1, class_weight=class_weights, max_depth=5)
forest.fit(X, y)

# define Boruta feature selection method
# percentage was set to 80 as the threshold for comparison between shadow and real features
# to allow for less drastic feature selection (way too many variables were being removed)
feat_selector = BorutaPy(forest, n_estimators='auto', verbose=2, random_state=1, perc=80)

# find all relevant features
feat_selector.fit(X, y)

feature_ranks = list(zip(df_.columns,
                         feat_selector.ranking_,
                         feat_selector.support_,
                         feat_selector.support_weak_))

# iterate through and print out the results
boruta = defaultdict(list)
for feat in feature_ranks:
    boruta['Feature'].append(feat[0])
    boruta['Rank'].append(feat[1])
    boruta['Keep'].append(feat[2])


# dataframe with ranks
boruta = pd.DataFrame(boruta).set_index('Feature')
```

The dataframe generated above returns information on rank of importance and whether Boruta's method would keep or exclude the feature. The following 17 features were kept:

```
['mean_sent', 'STOCH_slowk', 'STOCH_slowd', 'MACD','MACD_hist',
'CCI_5', 'CCI_10', 'CCI_21', 'CCI_50', 'MOM_5', 'MOM_10','MOM_21',
'RSI_5', 'RSI_10', 'RSI_21', 'RSI_50']
```

Finally, the correlation matrix between the selected features reiterates the selection, the features are considerably correlated to Bitcoin's returns:

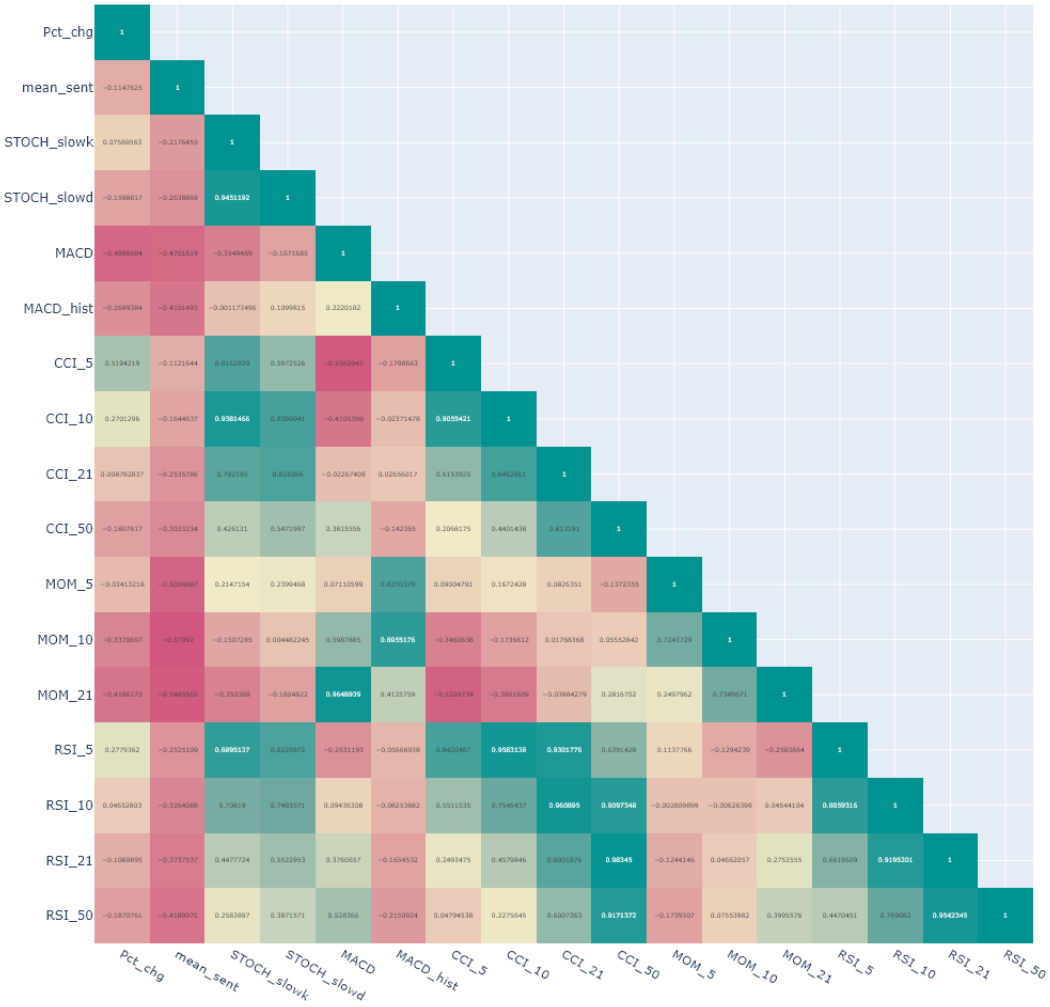

Figure 21: Correlation matrix of Selected Features.

4 MODEL TRAINING AND RESULTS

The aim was to predict the *target* (Bitcoin returns up/down movements) with the selected features. For further details, consult the full code in the notebook.

4.1 LSTM architecture

```
model.add(LSTM(units=params['u1'], input_shape = (win_length, num_features), \
return_sequences=True))
# Dropout layer
model.add(Dropout(params['dropout']))
#Add second layer LSTM
model.add(LSTM(units=params['u2'], return_sequences=False))
#Dropout layer
model.add(Dropout(params['dropout']))
#Add a Dense layer
model.add(Dense(params['dense_neurons'], activation = params['activation']))

#Add the output layer - output layer
model.add(Dense(1, activation = 'sigmoid'))#output layer
```

1. A first LSTM and input layer with dynamic dimension defined by window lenght and number of features
2. A Dropout layer to prevent overfitting (it randomly shuts down a couple of neurons to avoid to do so)
3. Another LSTM layer, stacked 2 layer LSTM tend to provide enough complexity in most of the cases
4. Yet another dropout layer to prevent overfitting and regularize information
5. A Dense, output layer, fully connected to previous neurons to condense information before the output
6. Finally, a Dense output layer, wherein the activation function is a Sigmoid due to the problem at hand: 0 or 1 (up or down) dependent variable classification.

4.2 Hyperparameter Tuning and Compiling

The following parameters were tuned using Talos to avoid arbitrary selection.

```
params = {'lr': [1e-2,1e-3, 1e-4],
    'u1': [128,256],
    'u2': [128,256],
    'dropout': [0.3, 0.5],
    'batch_size': [32,64, 256],
    'epochs': [100],
    'optimizer': ['Adam'],
    'activation': ['relu','elu'],
    'dense_neurons': [32, 64],
    'window': [21,50,200],
    }
```

Each permutation of the values above for each hyperparameter was tested (with the time restriction, Talos was constrained to return only 20% of possible permutations):

1. lr: Learning Rate, the rate at which the model learns
2. u1 and u2: The number of nodes in the first and second LSTM layers
3. dropout: The number of nodes in the first and second LSTM layers
4. batch_size: Number of training samples to work through before the model's internal weights are updated
5. epochs: Number times that the learning algorithm will work through the entire training data (although this impacts the result, it was kept constant in 100 due to time restrictions)
6. optimizer: Many alternatives were tested, but in the end this project was restricted to Adam optimizer, an enhanced SGD, due to its efficiency, speed and good performance on large datasets
7. activation: Relu and Elu were both tested
8. dense_neurons: Number of nodes in the last, dense, hidden layer
9. window: Lookback or time window considered (how many days are being used to make the next day's prediction)

A couple of metric functions (Recall, Precision, Specificity and F1) were manually defined to be computed on each iteration, they were defined as follows:

```
    # Defining Recall, Precision, Specificity and F1 metrics:
#(https://medium.com/analytics-vidhya/custom-metrics-for-keras-tensorflow-ae7036654e05)
def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall_keras = true_positives / (possible_positives + K.epsilon())
    return recall_keras


def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision_keras = true_positives / (predicted_positives + K.epsilon())
    return precision_keras


def specificity(y_true, y_pred):
    tn = K.sum(K.round(K.clip((1 - y_true) * (1 - y_pred), 0, 1)))
    fp = K.sum(K.round(K.clip((1 - y_true) * y_pred, 0, 1)))
    return tn / (tn + fp + K.epsilon())


def f1(y_true, y_pred):
    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    return 2 * ((p * r) / (p + r + K.epsilon()))
```

Additionally, in order to avoid that the Binary Cross Entropy loss function's gradient descent taking sharp turns or diverging on later iterations, an exponential decay of -0.1 per epoch on the learning rate was set as follows (to be used as a callback when compiling the model):

```
# This function keeps the initial learning rate for the first ten epochs
# and decreases it exponentially after that.
def scheduler(epoch, lr):
    #rampup epochs
    if epoch<=10:
        return lr
    return lr * tf.math.exp(-0.1*epoch)
```

Finally, in order to avoid overfitting, an Early Stopping was set to stop after 10 consecutive epochs without reduction of the loss in the validation set.

```
early_stopping = EarlyStopping(monitor='val_loss',patience = 10, mode='min',\
restore_best_weights=True)
```

The Talos' tuning was set to return only 20% of all possible permutations due to time constraints.

```
talos.Scan(x = x_train, y = y_train, x_val = x_test, y_val = y_test,
params = params, model = lstm_model,experiment_name = 'lstm_model_val2',
fraction_limit = 0.2) # this yields only 20\% permutations
```

To check the whole code, please go to: https://github.com/Yamamuen/CQF_final_project/blob/main/DL.ipynb.

4.3 Best Model

The hyperparameter tuning for 20% of the possible permutations returned 172 different models.

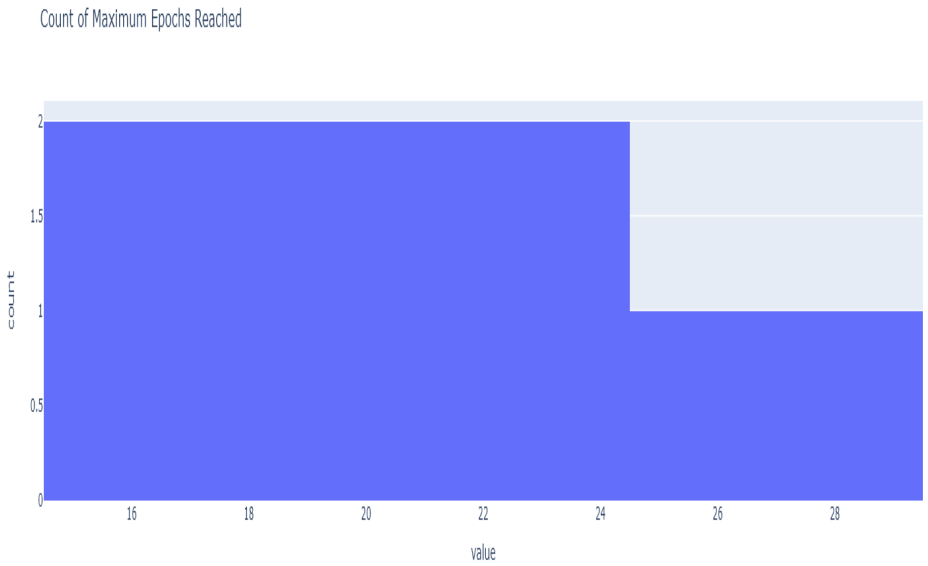

Figure 22: Histogram of maximum epochs reached per permutation.

No model went over 30 epochs, even though the epochs were set to 100. Clearly, either the early stop was too stringent or the exponential decay was too intense, making the gradient descent stop too early. For all the computed models, the Validation Binary Cross Entropy loss stayed in the 0.68-0.69 range. Accuracy is almost normally distributed but also highly concentrated.

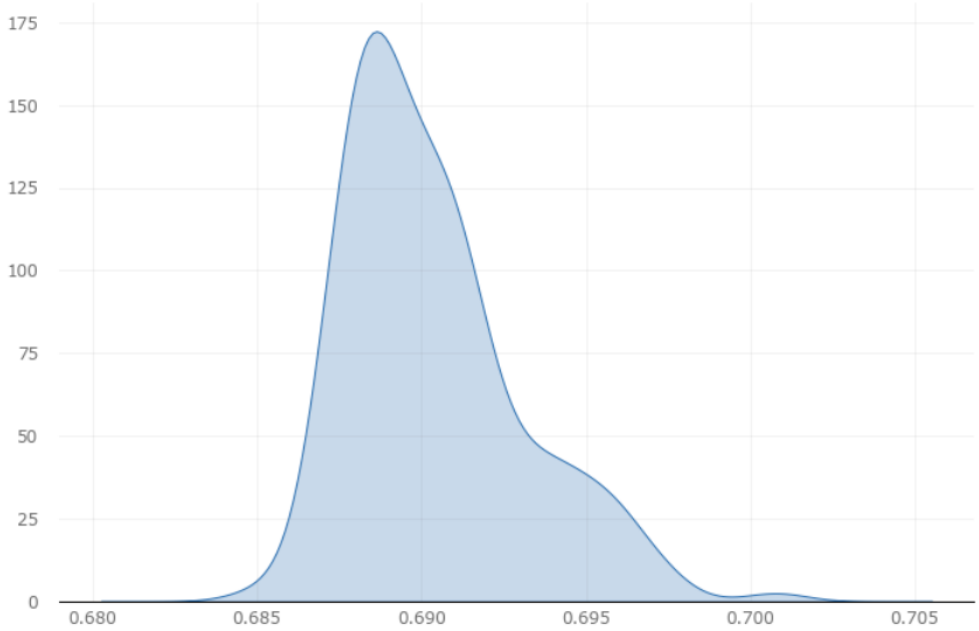

Figure 23: Validation Cross Entropy Loss Distribution.

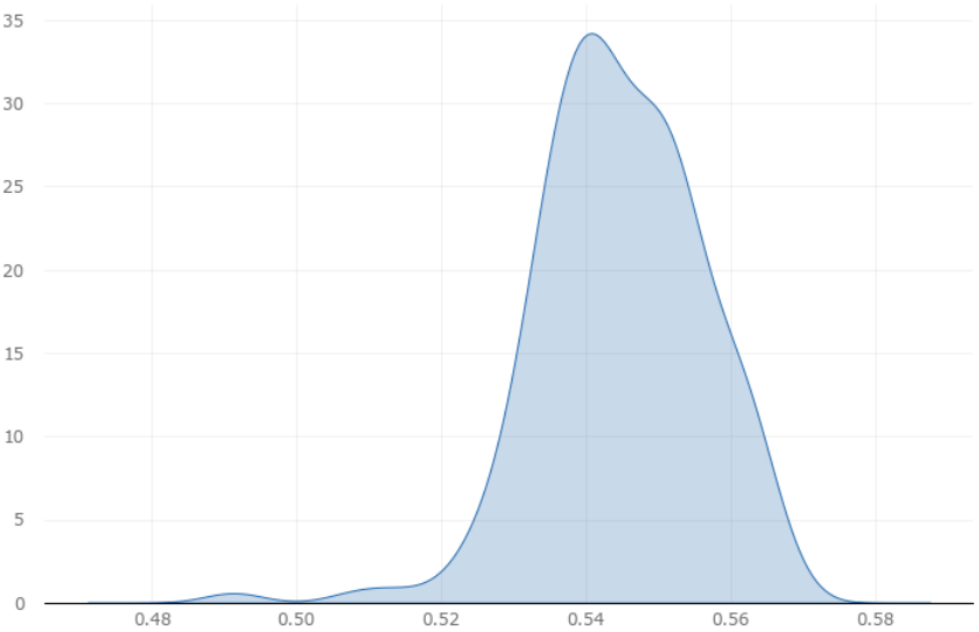

Figure 24: Validation Cross Entropy Loss Distribution.

The heatmap on the next page shows that there is, weirdly, no clear relation between the tuned hyperparameters and the metrics considered. Bearing in mind the shortfall engendered by many restrictions and the considerably high minimum value for the validation loss, these were the best possible hyperparameters produced after running 20% of the total possible permutations:

```
{'lr': 0.0001,'u1': 128,'u2': 256,'dropout': 0.5,'batch_size': 32,
'epochs': 100,'optimizer': 'Adam','activation': 'relu','dense_neurons': 64,'window': 200}
```

The best model produced has 128 neurons in the first layer, 256 in the second and 64 in the last, dense layer with ReLU activation. Besides, the dropout rates are of 0.5, the batch size is 32 and 200 previous days are used to classify the next day price movement. The learning rate is the smallest between the alternatives provided.

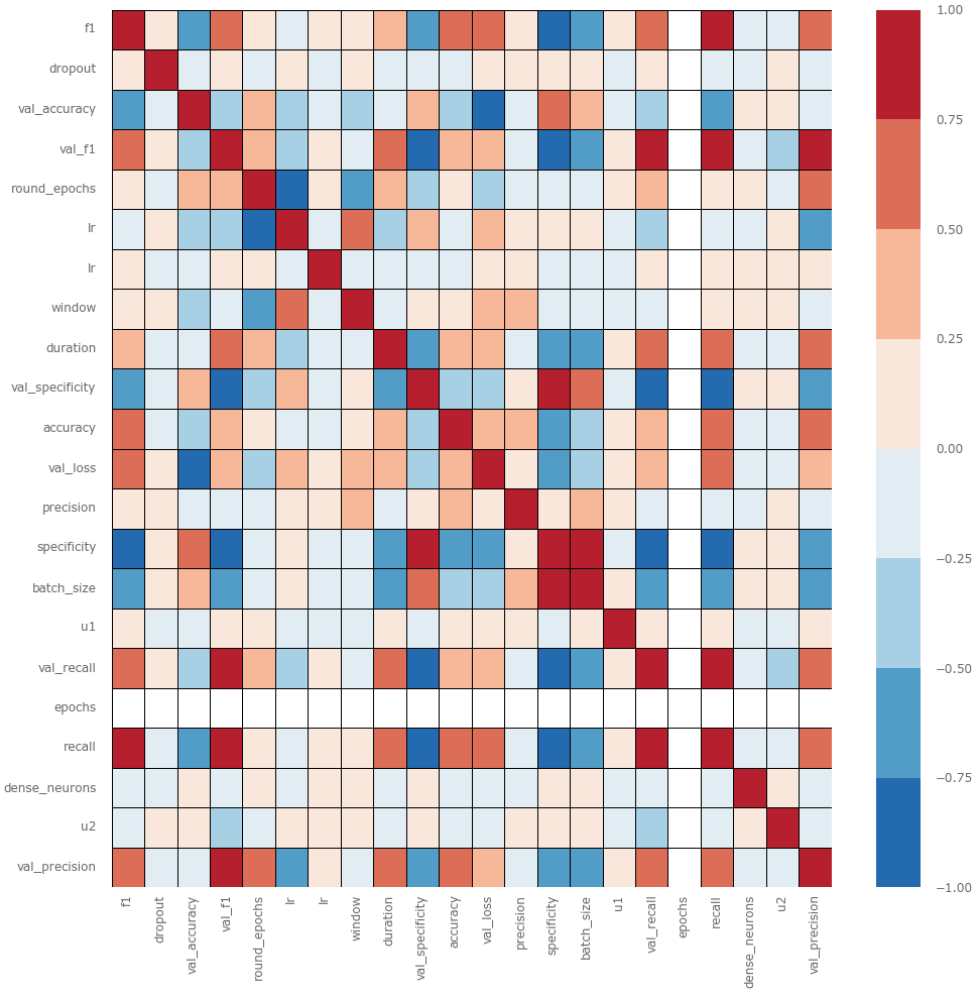

Figure 25: Correlations: Metrics x Hyperparameters.

5 RESULTS

5.1 Main Model - Classification

Using the optimal parameters, a slightly different model is trained: for longer epochs (200) and with increased early stopping (stops only after 30 stale periods) tolerance. Over the training, the metrics evolved as follows:

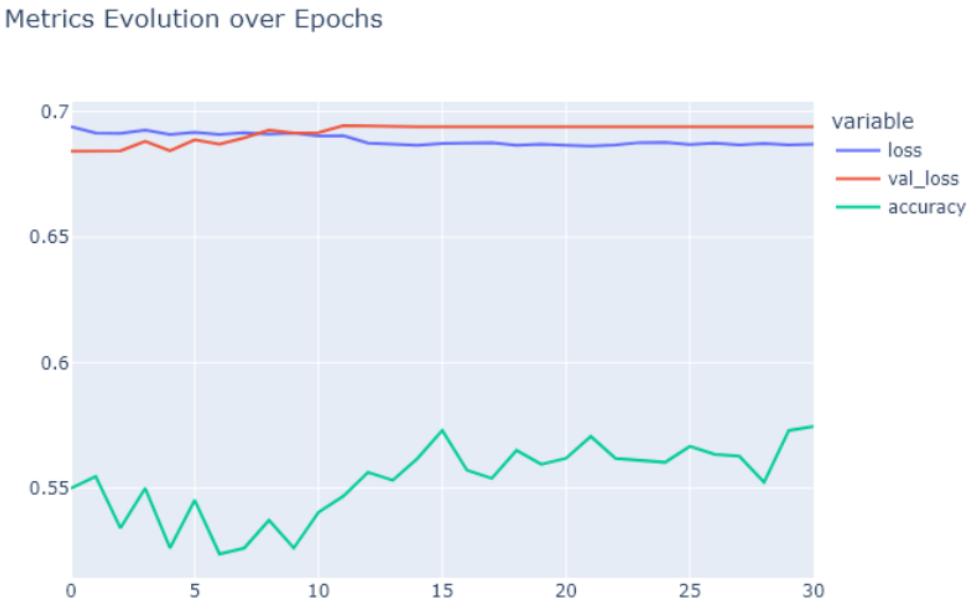

Figure 26: Metrics evolution over time.

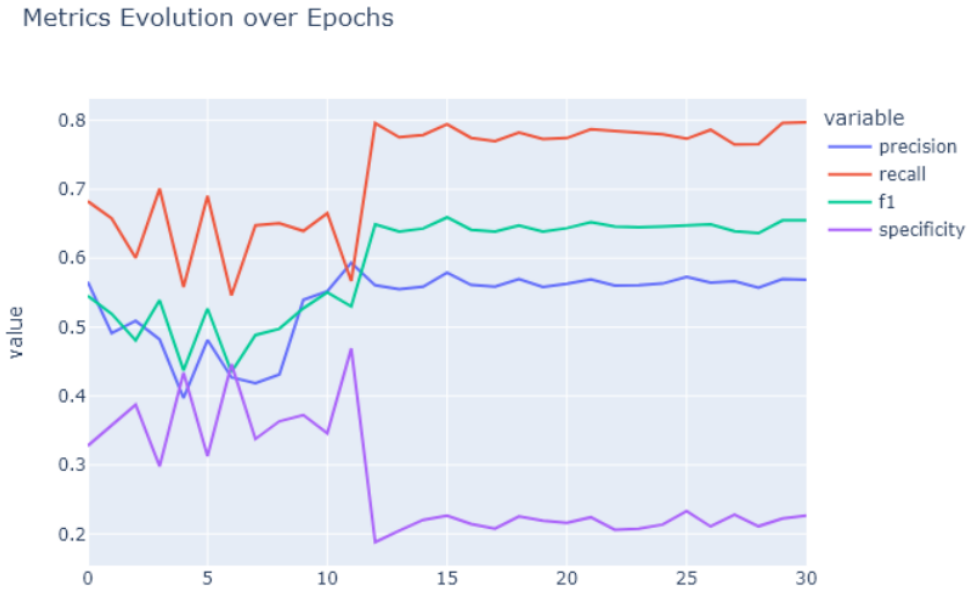

Figure 27: Metrics evolution over time.

Unfortunately, increasing the number of epochs caused overfitting: the validation loss starts to increase whilst the loss decreases, with a subsequent increasing accuracy metric. Furthermore, even with the improvement of Precision, Recall and F1-score over the epochs, it came at the cost of Specificity, the true negative rate, worsening over time.

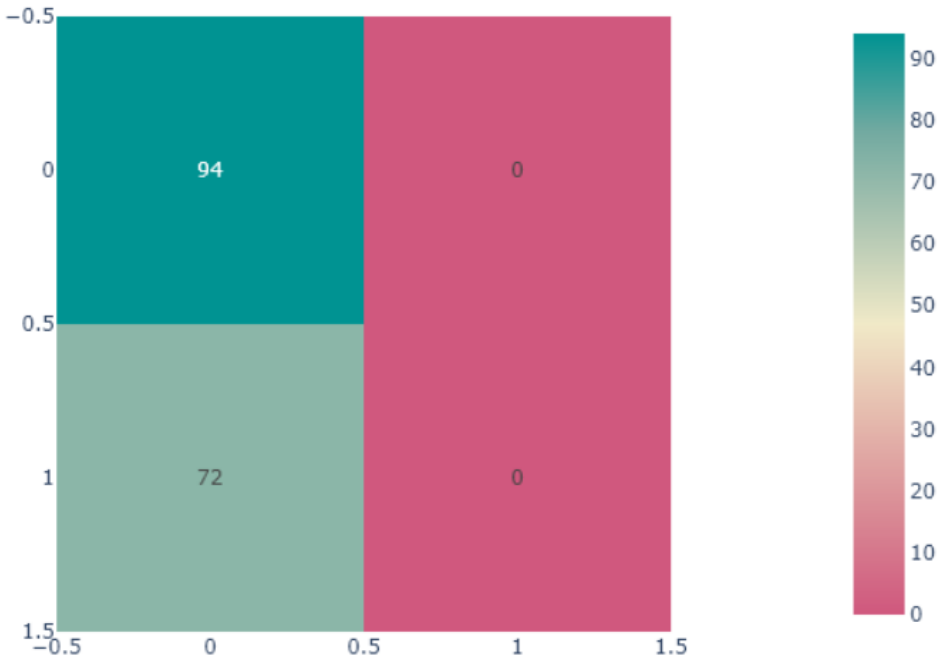

Figure 28: Confusion matrix.

The confusion matrix shows that the model only predicted downward moves.

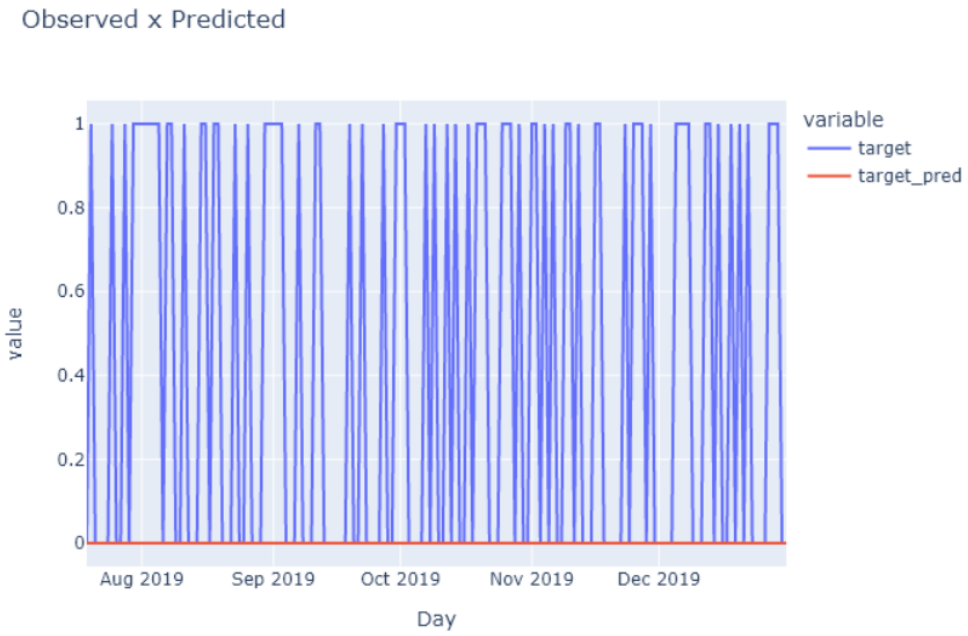

Figure 29: Confusion matrix.

Apart from not converging, the trained model also failed to predict every single upward movement on the test set. The accuracy is over 50% only due to it always predicting movements as downwards in a test set where up and down movements are somewhat distributed. Likely to be either an architecture or data processing issue, although thus far its cause could not be pinpointed.

5.2 Alternative Model - Regression

Out of curiosity, another model (regressor) was created and tested based solely on Bitcoin Close price (Y) and Aggregated Daily Sum of Weighted Sentiments (X). This was done by trial and error (choosing between weekly time windows and a couple of batch sizes), without any sophisticated hyperparameter tuning, but still yielded better results. The following architecture and hyperparameters were used and the plot shows how it performed against the test set.

Configuração	MSE	MAE	MAPE
Batches:8, Janela:28	0.028412	0.001287	0.028412
Batches:8, Janela:14	0.029148	0.001342	0.029148
Batches:4, Janela:28	0.030221	0.001490	0.030221

Table 2: Best batch and window (janela) configurations.

The best configuration above yielded extremely low errors, with a Mean Average Percentual Error of 2.8%. Using batches = 8 and window lenght = 28 days, early stop set to quit after 3 trials without reducing validation loss and an Adam optimizer with a 0.001 learning rate coupled with a fixed exponential decay of -0.1 (this time it does not depend on epochs), the following code was executed:

```
# best config
win_length = 28
batch_size = 8

num_features = x_train.shape[1]


train_generator = TimeseriesGenerator(x_train, y_train, length= win_length,\
sampling_rate = 1, batch_size= batch_size)
test_generator = TimeseriesGenerator(x_test, y_test, length= win_length, \
sampling_rate = 1, batch_size= batch_size)

# Architecture
model = Sequential()
model.add(LSTM(128, input_shape = (win_length, num_features), return_sequences = True))
model.add(LeakyReLU(alpha=0.5))
model.add(LSTM(128, return_sequences=True))
model.add(LeakyReLU(alpha=0.5))
model.add(Dropout(0.3))
model.add(LSTM(64, return_sequences=False))
model.add(Dropout(0.3))
model.add(Dense(1)) #output layer

# Early stop
early_stopping = EarlyStopping(monitor='val_loss',patience = 3, mode='min')

# Exponential decay
def scheduler(epoch, lr):
  return np.clip(lr * tf.math.exp(-0.1), 0.000001, 0.001)
```

```
callbacklr = tf.keras.callbacks.LearningRateScheduler(scheduler)

model.compile(loss = 'mae',
              optimizer = tf.optimizers.Adam(learning_rate=0.001),
              metrics=['mse','mape'])


set_seed(42)
history = model.fit(train_generator, epochs = 200, validation_data = test_generator,
                                shuffle=False, callbacks = [early_stopping, callbacklr])
```

The relevant regression metrics behaved over the epochs as follows:

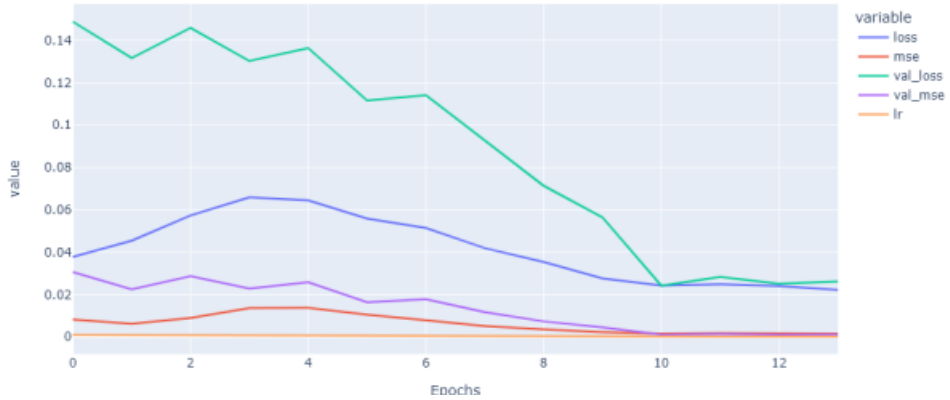

Figure 30: Metric evolutions over time.

The predicted x observed values in the test period (2019) were:

Figure 31: Predicted x Observed for LSTM regressor.

6 CONCLUSION

Time was limited (personal issues) and processing times were huge, even when setting Tensor Flow to run on GPU or on the Google Colab's Pro GPUs. That said, many arbitrary limitations were set, such as restricting the hyperparameter tuning to 20% of total possible permutations and testing between a fixed, small amount of values per hyperparameter. Furthermore, as a lot of time was spent in trying to bring Reddit BTC sentiment data (it was not readily available and was done from scratch), in order to make the task achievable, only one asset was analysed.

Even though no direct baseline models were documented here, besides the alternative configuration from the Talos tuning phase, many different architectures were devised throughout the course of this project. All of which also failed to converge bellow the 0.68 Binary Cross Entropy log loss threshold. Either the features are inadequate to explain Bitcoin up / down movements or the considered parameters were far from ideal, as many attempts were made with different architectures. Finally, although the classification did not work, it could be argued that simply letting the gradient descent run for longer epochs could generate a better fit. Besides, as per the Adam optimizer logic, it should not require any sort of decay, which means the somewhat drastic exponential decay defined could be discarded altogether.

Interestingly, when using the alternative model which, instead of attempting to classify BTC's direction, tried to predict its closing price, far more impressive results were obtained. An important caveat is that only the aggregated daily sums of weighted sentiments (which considers volume and polarity simultaneously) was used as a feature. The use of this single variable or the LeakyRelu activation layers (which solve ReLu's vanishing gradient problem) could explain it, had I not attempted the exact same architecture for the classification problem by changing the final dense layer's activation to a sigmoid (also produced poor results).

Although the classification models performed poorly, the alternative regression model provided impressive results, closely following the trends with a small noticeable delay. Perhaps a regression model with outputs adapted to predict up and downward moves would perform even better. There is a lot of room for improvement (specially in the classifier), but this at least prove that Reddit, at least during the time span considered, as a niche hub for crypto enthusiasts, reflected overall market sentiment and conditions fairly well. Nowadays, it may not have fared so well, as the cryptocurrency dynamics changed, with big players joining in on the fun.

REFERENCES

[1] Eugene F. FAMA. Efficient capital markets: Ii. *The Journal of Finance*, 46(5):1575–1617, 1991.

[2] Eugene F. Fama, Lawrence Fisher, Michael C. Jensen, and Richard Roll. The adjustment of stock prices to new information. *International Economic Review*, 10(1):1–21, 1969.

[3] Daniel Kahneman and Amos Tversky. Choices, values, and frames. In *Handbook of the fundamentals of financial decision making: Part I*, pages 269–278. World Scientific, 2013.

[4] Twitter. Twitter investor fact sheet q12021. Available in: https://s22.q4cdn.com/826641620/files/doc_financials/2021/q1/Q1'21_InvestorFactSheet.pdf. Accessed on: 12 de Jul. 2021, 2021.

[5] Brian Dean. Reddit user and growth stats. Available in: https://backlinko.com/reddit-users. Accessed on: 12 de Jul. 2021, 2021.

[6] CNN. Wall street reddit hedge funds. Available in: https://edition.cnn.com/2021/02/03/investing/wall-street-reddit-hedge-funds/index.html. Accessed on: 12 de Jul. 2021, 2021.

[7] Ron Shevlin. How elon musk moves the price of bitcoin with his twitter activity. Available in: https://www.forbes.com/sites/ronshevlin/2021/02/21/how-elon-musk-moves-the-price-of-bitcoin-with-his-twitter-activity/?sh=750ef005d27b. Accessed on: 12 de Jul. 2021, 2021.

[8] Emil Sayegh. Losing touch with reality – a gamestop lesson. Available in: https://www.forbes.com/sites/emilsayegh/2021/03/09/losing-touch-with-reality--a-gamestop-lesson/?sh=4c74871735a5. Accessed on: 12 de Jul. 2021, 2021.

[9] Kit Smith. 60 incredible and interesting twitter stats and statistics. Available in: https://www.brandwatch.com/blog/twitter-stats-and-statistics/. Accessed on: 12 de Jul. 2021, 2020.

[10] Thien Hai Nguyen and Kiyoaki Shirai. Topic modeling based sentiment analysis on social media for stock market prediction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1354–1364, 2015.

[11] Jethin Abraham, Daniel Higdon, John Nelson, and Juan Ibarra. Cryptocurrency price prediction using tweet volumes and sentiment analysis. *SMU Data Science Review*, 1(3):1, 2018.

[12] Shubhankar Mohapatra, Nauman Ahmed, and Paulo Alencar. Kryptooracle: A real-time cryptocurrency price prediction platform using twitter sentiments. pages 5544–5551, 12 2019.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] C.J. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. 01 2015.

[16] Dennis Yang and Qiang Zhang. Drift-independent volatility estimation based on high, low, open, and close prices. *The Journal of Business*, 73(3):477–492, 2000.

[17] Steven Walczak. An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of management information systems*, 17(4):203–222, 2001.

[18] Dev Shah, Haruna Isah, and Farhana Zulkernine. Predicting the effects of news sentiments on the stock market. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4705–4708. IEEE, 2018.

[19] Hong Kee Sul, Alan R Dennis, and Lingyao Yuan. Trading on twitter: Using social media sentiment to predict stock returns. *Decision Sciences*, 48(3):454–488, 2017.

[20] Marc Velay and Fabrice Daniel. Stock chart pattern recognition with deep learning. *arXiv preprint arXiv:1808.00418*, 2018.

[21] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[22] Erikson Júlio de Aguiar, Bruno S Faiçal, Jó Ueyama, Glauco Carlos Silva, and André Menolli. Análise de sentimento em redes sociais para a língua portuguesa utilizando algoritmos de classificação. In *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC, 2018.

[23] Joana Matos Dias, Katarzyna Wegrzyn-Wolska, Imen Rached, Horacio González-Vélez, Roman Senkerik, Claudia Pop, Tudor Cioara, Ioan Salomie, and Andrea Bracciali. Forecasting cryptocurrency value by sentiment analysis: An hpc-oriented survey of the state-of-the-art in the cloud era. *High-Performance Modelling and Simulation for Big Data Applications*, page 325, 2019.

[24] Svitlana Galeshchuk, Oleksandra Vasylchyshyn, and Andriy Krysovatyy. Bitcoin response to twitter sentiments. In *CEUR Workshop Proceedings*, pages 160–168, 2018.

[25] Tom M Mitchell et al. Machine learning. 1997.

[26] Ana Carolina Lorena, João Gama, and Katti Faceli. *Inteligência Artificial: Uma abordagem de aprendizado de máquina*. Grupo Gen-LTC, 2000.

[27] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[28] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Machine Learning - A First Course for Engineers and Scientists*. 2021.

[29] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*, volume 72. Springer, 2015.

[30] Evita Stenqvist and Jacob Lönnö. Predicting bitcoin price fluctuation with twitter sentiment analysis, 2017.

[31] Jeff John Roberts. Big bitcoin crashes: What we learned. *Fortune, available at: http://fortune. com/2017/09/18/bitcoin-crash-history/(18 September)*, 2017.

[32] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[33] Raymond J Dolan. Emotion, cognition, and behavior. *science*, 298(5596):1191–1194, 2002.

[34] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326, 2010.

[35] Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. Twitter sentiment analysis: The good the bad and the omg! In *Fifth International AAAI conference on weblogs and social media*, 2011.

[36] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8, 2011.

[37] Ziaul Haque Munim, Mohammad Hassan Shakil, and Ilan Alon. Next-day bitcoin price forecast. *Journal of Risk and Financial Management*, 12(2):103, 2019.

[38] Jigsaw Research. News consumption in the uk: 2020. Available in: https://www.ofcom.org.uk/__data/assets/pdf_file/0013/201316/news-consumption-2020-report.pdf. Accessed on: 12 de Jul. 2021, 2020.

[39] Sarah Perez. Majority of consumers use social networks to inform buying decisions, says study. Available in: https://archive.nytimes.com/www.nytimes.com/external/readwriteweb/2010/07/26/26readwriteweb-majority-of-consumers-use-social-networks-t-90514.html. Accessed on: 12 de Jul. 2021, 2010.

[40] IBM Cloud Education. Machine learning. Available in: https://www.ibm.com/cloud/learn/machine-learning. Accessed on: 12 de Jul. 2021, 2020.

[41] Ahmet Özlü. Long short term memory (lstm) networks in a nutshell. Available in: https://ahmetozlu.medium.com/long-short-term-memory-lstm-networks-in-a-nutshell-363cd470ccac. Accessed on: 12 de Jul. 2021, 2020.

[42] MacKenzie Sigalos. Bitcoin has become a lifeline for sex workers, like this former nurse who made $1.3 million last year. Available in: https://www.cnbc.com/2022/02/05/bitcoin-a-lifeline-for-sex-workers-like-ex-nurse-making-1point3-million.html. Accessed on 24 Mar. 2022, 2022.

[43] Tom Hale. How much data does the world generate every minute? Available in: https://www.iflscience.com/technology/how-much-data-does-the-world-generate-every-minute/. Accessed on 16 Jul. 2021, 2017.