Probabilistic Time Series Analysis

Final Project - Fall 2022

Bitcoin Price Prediction using Long Short-Term Memory Model (LSTM) and Hidden Markov Model (HMM)

Almadi Shiryayev

Abstract

The topic of this research paper is "Bitcoin Prediction using LSTM and HMM." In recent years, the popularity and adoption of cryptocurrencies, particularly Bitcoin, have increased significantly. As a result, it has become increasingly important to be able to accurately predict the future value of Bitcoin. One approach to predicting the value of Bitcoin is to use machine learning techniques, such as Long Short-Term Memory (LSTM) and Hidden Markov Models (HMM). These techniques have been successful in a variety of predictive tasks and have the potential to provide accurate predictions for Bitcoin. In this research paper, we propose a new approach for predicting the value of Bitcoin using LSTM and HMM. We evaluate the performance of our approach using historical data and compare it to other commonly used methods. The results of our analysis provide insight into the effectiveness of using LSTM and HMM for predicting the value of Bitcoin and may inform future research in this area. Overall, this research aims to contribute to the growing body of knowledge on the use of machine learning techniques for predicting the value of Bitcoin.

Contents

1 Introduction

Bitcoin, the world's first and most widely used decentralized digital currency, has seen its value fluctuate significantly over the years. Understanding and predicting its price movements can be of great value to investors, traders, and other stakeholders. In this essay, we will explore two methods for predicting Bitcoin price: Long Short-Term Memory (LSTM) neural networks and hidden Markov models (HMM).

LSTM neural networks are a type of deep learning model that are well-suited for analyzing time series data. They are able to remember and learn from long-term dependencies in the data, making them effective at predicting future values based on past trends.

On the other hand, HMMs are probabilistic models that are used to analyze sequential data. They are based on the idea of a hidden state, which is not directly observed but can be inferred from the observations. HMMs are often used to model financial time series data, as they can capture the underlying patterns and trends in the data.

In this essay, we will compare the performance of LSTM and HMM models on Bitcoin price prediction, and discuss their strengths and limitations. We will also explore the factors that may affect the accuracy of these models and ways to improve their performance.

2 Related Work

While doing a research on the topic of Bitcoin Price Prediction using LSTM, I found a paper which used LSTM but in the stock market [1]. I read all of the code to understand the logic behind the algorithm and successfully implemented the knowledge in my code. I implemented LSTM algorithm and experimented with different numbers of epochs. Additionally, we improved the Bayesian information criteria (BIC) calculation function by writing a proper formula in Python.

Furthermore, I also found another paper but with HMM implementation for stock prediction [2]. The reason for that is also to understand the logic behind HMM when predicting time series. I made some changes in methodology and implemented Bayesian Information Criterion (BIC).

As for the evaluation metric I used the Mean Absolute Percentage Error (MAPE) evaluation metric (I will explain why I chose it below), which helps me to evaluate the accuracy of both LSTM and HMM, and it also helps to evaluate different aspects like activation function, number of epochs etc., because MAPE was calculated after each change of these aspects to find the best conditions for the algorithms.

3 Problem definition and algorithms

3.1 Task

The main task of this research is to get precise predictions of Bitcoin price using Long Short-Term Memory Model (LSTM) and Hidden Markov Model (HMM). In both cases the first step is to gather a dataset of historical bitcoin price data. I will then need to preprocess the data by cleaning and formatting it in a way that is suitable for training the LSTM and HMM. This involves formatting the data, normalizing the data, removing missing values, and splitting the data into training and testing sets.

The input to the models is a sequence of Bitcoin prices, typically in the form of a NumPy array or a Pandas series. Further, these inputs will train our model and test it on testing set.

The output of the both models is a sequence of predicted Bitcoin prices, also in the form of a numpy array. The predicted prices will correspond to the same time frame as the input prices.

3.2 Algorithm

To achieve the main goal I have to define the LSTM model architecture. The LSTM model consists of a series of layers that process the input data and produce the output prediction. You will need to define the architecture of the LSTM model, including the number of layers, the number of units in each layer, and any additional hyperparameters such as the dropout rate. It is generally a good idea to start with a relatively simple model architecture and increase the complexity if necessary. The model example is shown on the Figure 1 where the LSTM used in that example has 3 LSTM layers and 1 Dense layer. The code for such LSTM is [3]:

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np

model = Sequential()
model.add(LSTM(32, return_sequences=True,
              input_shape=(timesteps, data_dim))) # returns a sequence of vectors of
                  dimension 32
model.add(LSTM(32, return_sequences=True)) # returns a sequence of vectors of
    dimension 32
model.add(LSTM(32)) # return a single vector of dimension 32
model.add(Dense(10, activation='softmax'))
```

And for the HMM part an example is shown below. We see how there are two split datasets one of which fits into model to teach algorithm and other is used for predictions.

```
import numpy as np
from hmmlearn import hmm

# Define the model
model = hmm.GaussianHMM(n_components=3)

# Train the model on the data
X = np.array([[1,2,3], [2,3,4], [3,4,5], [4,5,6]])
model.fit(X)

# Use the model to make predictions on new data
X_test = np.array([[5,6,7], [6,7,8], [7,8,9], [8,9,10]])
predictions = model.predict(X_test)

print(predictions)
```

4 Experimental evaluation

4.1 Data

At first we read the data:

```
data = pd.read_csv('BTC.csv')
data.head()
```

First two things that I instantly did are renaming "Price" column to "Close" name to make it easier to comprehend and getting rid of nan, empty values in the dataset:

```
data.rename(columns = {'Price':'Close'}, inplace = True)
data.dropna(inplace=True)
```

After that I noticed that the data type of the values in the dataframe is string, that's why I had to convert them to float so that it will work in algorithms:

```
data['Close'] = data['Close'].str.replace(',', '').astype("float32")
data['Close'] = pd.to_numeric(data['Close'])
```

Scaling the data is also an important part because it helps to ensure that all features are on the same scale, which can improve the performance of the model. For example, if one feature has a range of 0-1 and another feature has a range of 0-1000000, the model may place more weight on the feature with the larger range, even if both features are equally important. Scaling the data helps to prevent this type of imbalance. It also can help to reduce the effect of noise in the data. For example, if one feature has a large standard deviation compared to the other features, it may dominate the model's predictions even if it is not particularly relevant. Scaling the data can help to reduce the impact of such features. The code is shown below:

```
target = data['Close']
scaler=MinMaxScaler(feature_range=(0,1))
target=scaler.fit_transform(np.array(target))
print(target[:5])

[[0.00078306]
 [0.04282314]
 [0.04949746]
 [0.05735445]
 [0.05417943]]
```

4.2 Methodology

In this experiment I used LSTM and HMM models to predict bitcoin prices. In both cases I split dataset into training and testing sets:

```
X_train, X_test, y_train, y_test = train_test_split(target, data['Date'],
    random_state=0, train_size = .75)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

((502, 1), (168, 1), (502,), (168,))
```

For the LSTM part, I used two models one of which contains 1 Dense Layer and the other one contains 4 Dense Layers to check how the number of layers affect the accuracy of the model:

```
# 1 Dense Layer
model=Sequential()
model.add(LSTM(10,input_shape=(None,1),activation="relu"))
model.add(Dense(1))
model.compile(loss="mean_squared_error",optimizer="adam")
history =
    model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=200,batch_size=32,verbose=1)

# 4 Dense Layers
model=Sequential()
model.add(LSTM(10,input_shape=(None,1),activation="relu"))
model.add(Dense(24))
model.add(Dense(18))
model.add(Dense(22))
model.add(Dense(1))
model.compile(loss="mean_squared_error",optimizer="adam")
history =
    model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=200,batch_size=32,verbose=1)
```

After that I simply use model.predict function and calculate inverse of a scaled data to get the Bitcoin price predictions. The code is shown below:

```
train_data_prediction = model.predict(X_train)
test_data_prediction = model.predict(X_test)

train_data_predict = scaler.inverse_transform(train_predict)
test_data_predict = scaler.inverse_transform(test_predict)
y_train_initial = scaler.inverse_transform(y_train.reshape(-1,1))
y_test_intial = scaler.inverse_transform(y_test.reshape(-1,1))
```

Bayesian Information Criterion (BIC) was derived according to the formula where LL is the log-likelihood of the model, N is the number of examples in the training dataset, and n is the number of parameters in the model. [4] It is needed to find optimal number of hidden states in HMM by choosing the model with lowest deviation in results between the data in train and test data:

$$BIC = -2 * LL + n * log(N) \tag{1}$$

```
def bic(model, X):
    n_samples, _ = X.shape
    logL = model.score(X)
    n_features = model.n_features
    n_components = model.n_components
    logN = np.log(n_samples)
    return -2 * logL + n_components * n_features * logN

# Train and score an HMM model with a different number of hidden states
train_data = dataset[:, :-1]
test_data = dataset[:, -1]

number_states_list = list(range(1, 16, 1))

bic_scores = []
for state in states_list:
    model = hmm.GaussianHMM(n_components=state)
    model.fit(train_data)
    bic_scores.append(bic(model, test_data))

# Select the model with the lowest BIC score
opt_states = np.argmin(bic_scores) + 1

print("Optimal number of hidden states:", opt_states)

Optimal number of hidden states: 15
```

For the HMM part I wrote a code for calculating the correct value of the number optimal hidden states. When calculating the prediction the first HMM is taking place to get initial transition matrix, starting probabilities, means and covariance. I used the first 200 observations for testing and used the rest of the observations for training the model. The main idea for predicting the next day bitcoin price is to calculate the log-likelihood of K previous observations and compare it with the log-likelihood of all the previous sub-sequences of the same size by shifting by one day in the direction of past data. Then identify a day in the past whose log-likelihood of its K previous observations is the closest to the sub-sequence whose next day's price is to be predicted.

Calculated predicted change is added to the dataset's 200 other data values which were not used for training. As a result we can get the predicted Bitcoin price that was calculated using the previous days and the probability of the next day. The code is shown below:

```
NUM_TEST = 200
predicted_btc_price_data = []
for idx in reversed(range(NUM_TEST)):

    train_data = dataset[idx + 1:,:]
    num_examples = train_data.shape[0]

    # Initial start to get initial tranformation matrices, start probability, initial
        means and covars.
    if idx == NUM_TEST - 1:
        model = hmm.GaussianHMM(n_components=opt_states,tol=0.0001, n_iter=NUM_ITERS,
            init_params='stmc')
    else:

    # Retuning the model by using the HMM paramters from the previous iterations as
        the prior

        model = hmm.GaussianHMM(n_components=opt_states, tol=0.0001, n_iter=NUM_ITERS,
            init_params='')
        model.transmat_ = transmat_retune_prior
        model.startprob_ = startprob_retune_prior
        model.means_ = means_retune_prior
        model.covars_ = covars_retune_prior

    # Fitting the input sequence (training data) into algorithm
    model.fit(np.flipud(train_data))

    transmat_retune_prior = model.transmat_
    startprob_retune_prior = model.startprob_
    means_retune_prior = model.means_
    covars_retune_prior = model.covars_
    iters = 1
    prev_likelihood = []

    initial_likelihood = model.score(np.flipud(train_data[0:K - 1, :]))

    while iters < num_examples / K - 1:
         prev_likelihood = np.append(prev_likelihood,
             model.score(np.flipud(train_data[iters:iters + K - 1, :])))
        iters = iters + 1
    likelihood_diff_idx = np.argmin(np.absolute( prev_likelihood - initial_likelihood))
    predicted_change = train_data[likelihood_diff_idx,:] -
        train_data[likelihood_diff_idx + 1,:]
    print(predicted_change, likelihood_diff_idx)
    predicted_btc_price_data = np.vstack((predicted_btc_price_data, dataset[idx + 1,:]
        + predicted_change))

mape = calc_mape(predicted_btc_price_data, np.flipud(dataset[range(200),:]))
```

As for the evaluation metric in order to evaluate models I chose to use the mean absolute percentage error (MAPE) [5]:

$$MAPE = \frac{1}{N} * \sum |\frac{actualvalue - forecastvalue}{actualvalue}|$$

There are many different evaluation metrics that can be used to assess the performance of machine learning models, and choosing the right metric depends on the specific characteristics of the data and the goals of the modeling task. In the case of using LSTM (Long Short-Term Memory) and HMM (Hidden Markov Model) models for predicting the price of bitcoin, I chose to use the mean absolute percentage error (MAPE) as the evaluation metric for several reasons.

First, MAPE is a widely-used metric for evaluating the accuracy of time series predictions, and it is particularly well-suited for tasks where the magnitude of the errors is important. In the case of predicting the price of bitcoin, the percentage error of a prediction can be much larger than the absolute error, depending on the current price of bitcoin and the size of the prediction error. For example, a prediction error of $10 when the price of bitcoin is $100 would be a much larger percentage error than a prediction error of $10 when the price of bitcoin is $1,000. MAPE takes into account the percentage error of each prediction and averages them across the entire dataset, providing a more meaningful measure of the overall prediction accuracy.

Second, MAPE is relatively robust to the presence of outliers in the data. In the case of bitcoin price data, there can be large fluctuations in the price from one day to the next, and using a metric such as mean squared error (MSE) or root mean squared error (RMSE) could lead to distorted results due to the influence of these outliers. MAPE is less sensitive to the influence of outliers, making it a more reliable evaluation metric in this case.

4.3 Results

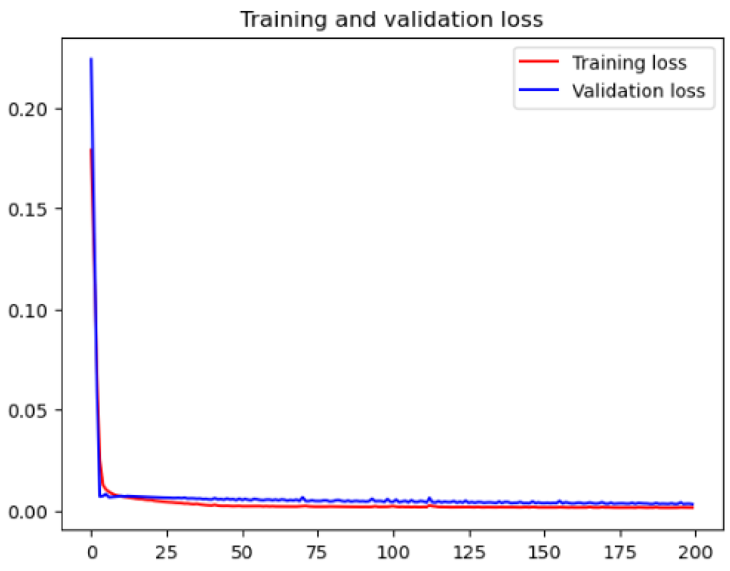

Figure 1: Training and validation loss of LSTM

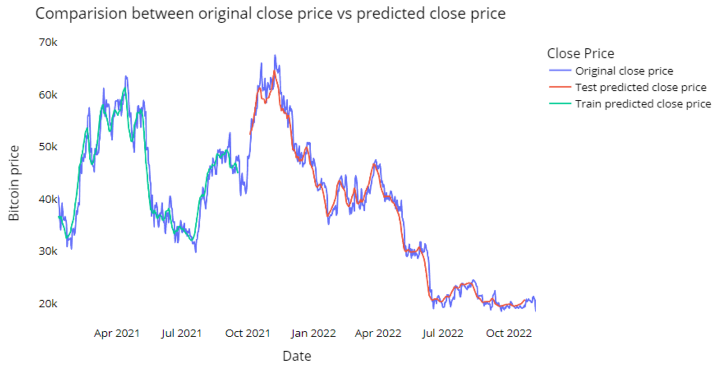

Figure 2: LSTM results showing original, test and train close prices

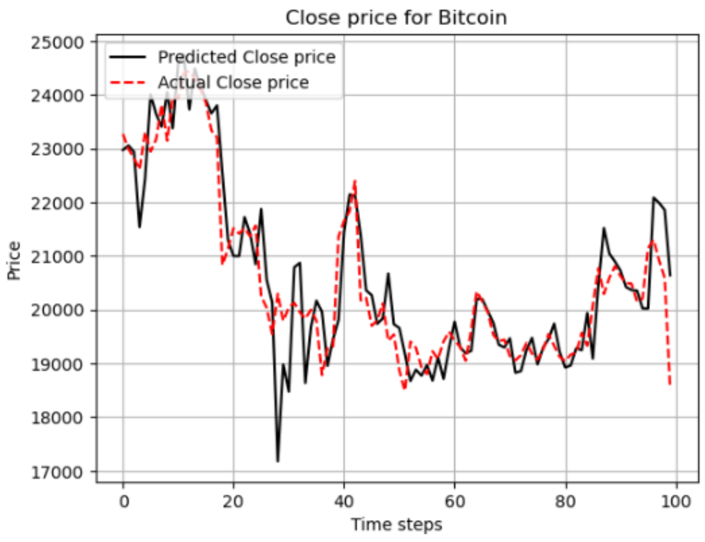

Figure 3: HMM results showing predicted and actual close prices

Model Name	MAPE
LSTM (1 Dense Layer)	0.036569227700739754
LSTM (4 Dense Layer)	0.036569227700739754
HMM	0.0276949669682594

Table 1: Mean Absolute Percentage (MAPE) results for different LSTM and HMM models

4.4 Discussion

The results in Table 1 above shows how far away (in %) are predictions made by HMM from the actual values they were aiming for. Thus, predictions made by HMM are on average 2.77% away from the actual values they were aiming for and predictions made by LSTM are on average 3.656923% away from the actual values they were aiming for. Thus, HMM performed better in prediction of bitcoin price since the model has lower mean absolute percentage error (MAPE) than LSTM model. However, both of the models performed relatively good. One of the reasons is that HMMs are particularly well-suited for modeling sequences of data that exhibit temporal dependencies, such as the price of a financial asset (in my case it is Bitcoin price) over time. If the price of Bitcoin exhibits strong temporal dependencies, an HMM might be able to capture these dependencies more effectively than an LSTM, leading to a lower MAPE.

In this particular case the target value was "Close" price of the bitcoin which was the only parameter used. Thus, the model is not complex in terms of learning. LSTM networks are generally more complex than HMMs, as they have multiple layers and units that can learn more sophisticated patterns in the data. If the data is not complex enough to justify the use of an

LSTM, an HMM might be able to achieve similar or better performance with a simpler model.

Besides, regarding the similarities, as was mentioned above both of the results are relatively good which means both are low. It is possible that the HMM and LSTM models were not trained correctly, leading to biased results. As there are only 670 entries in the dataset, it may result in underfitting to the training data, it might too good on the testing data, resulting in a lower MAPE.

5 Conclusion

It is difficult to make definitive statements about the relative performance of different machine learning algorithms, as the performance of a specific algorithm will depend on a wide range of factors, including the characteristics of the data, the specific problem being solved, and the implementation of the algorithm.

That being said, there are some situations where an HMM (hidden Markov model) may be a better choice than an LSTM (long short-term memory) network for predicting Bitcoin prices. One reason for this may be that HMMs are generally simpler and easier to implement than LSTMs, which can make them a more efficient and effective choice for certain tasks. Additionally, HMMs are well-suited for modeling sequences of observations with temporal dependencies, which may be important for predicting Bitcoin prices.

However, there are also situations where an LSTM network may be a better choice than an HMM. For example, LSTMs are particularly effective at modeling long-term dependencies, which may be important for predicting Bitcoin prices that are influenced by a wide range of factors over a long period of time. Besides, as I varied number of epochs in LSTM algorithm, the results showed positivity in the prediction since the MAPE value significantly decreased.

It's important to note that HMMs, like all machine learning models, can only make predictions based on the patterns they have learned from the training data. If there are significant changes in the market or other factors that are not reflected in the training data, the HMM may not be able to make accurate predictions. Therefore, it's important to regularly retrain the model on new data to ensure that it stays up to date with the latest market conditions.

Thus, further improvements include using a larger dataset and more parameters and factors so that algorithm can learn more and find more insights.

References

[1] F. SAYAH. Stock market analysis + prediction using lstm. [Online]. Available: https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm

[2] A. A. F. CHOWDHURY. Stock market - hmm trials - basic - 1. [Online]. Available: https://www.kaggle.com/code/fayadc/stock-market-hmm-trials-basic-1

[3] K. Documentation. Getting started with the keras sequential model. [Online]. Available: https://faroit.com/keras-docs/1.0.4/getting-started/sequential-model-guide/

[4] J. Brownlee. Probabilistic model selection with aic, bic, and mdl. [Online]. Available: https://towardsdatascience.com/bayesian-model-selection-as-a-feature-reduction-technique-70d75386eabc

[5] I. E. Team. Probabilistic model selection with aic, bic, and mdl. [Online]. Available: https://www.indeed.com/career-advice/career-development/what-is-mape#:~:text=Once%20you%20have%20the%20absolute,final%20result%20is%20the%20MAPE.