Build an algorithm

July 7, 2020

[88]:
```
from datetime import date, timedelta
import pandas as pd
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import autocorrelation_plot
from sklearn.linear_model import LinearRegression
from train_model import trainModel, getFeatures, standardScaler
from predict import predictNextDays
```

[2]:
```
STOCK_NAME = "BTC-USD"
# Using yahoo finance to get Stock data
df = yf.Ticker(STOCK_NAME).history(period="25mo").reset_index()
```

0.1 Functions of the algorithm

[3]:
```
def create_dtFeatures(dataframe):
    """
    Input:
        dataframe(pandas DataFrame):  BTC data
    Output:
        dataframe(pandas DataFrame): dataframe with the date time features
    """
    dataframe['Date'] = pd.to_datetime(df['Date'])
    dataframe['day'] = dataframe['Date'].dt.day
    dataframe['month'] = dataframe['Date'].dt.month
    dataframe['year'] = dataframe['Date'].dt.year
    return dataframe
```

[4]:
```
create_dtFeatures(df).head()
```

[4]:
```
        Date     Open     High      Low    Close      Volume  Dividends  \
0 2018-06-07  7650.82  7741.27  7650.82  7678.24  4485799936          0
1 2018-06-08  7685.14  7698.19  7558.40  7624.92  4227579904          0
2 2018-06-09  7632.52  7683.58  7531.98  7531.98  3845220096          0
3 2018-06-10  7499.55  7499.55  6709.07  6786.02  5804839936          0
4 2018-06-11  6799.29  6910.18  6706.63  6906.92  4745269760          0
```

```
   Stock Splits  day  month  year
0             0    7      6  2018
1             0    8      6  2018
2             0    9      6  2018
3             0   10      6  2018
4             0   11      6  2018
```

[5]:
```
def create_laggedFeatures(dataframe, feature, shift_value):
    """
    Input:
        dataframe(pandas DataFrame): BTC data
        feature(string): the dataframe column
        shift_value(int): the value of the lagged feature
    Output:
        dataframe(pandas DataFrame): dataframe with the lagged feature
    """
    dataframe[feature + '_lagged'] = dataframe[feature].shift(shift_value)
    return dataframe
```

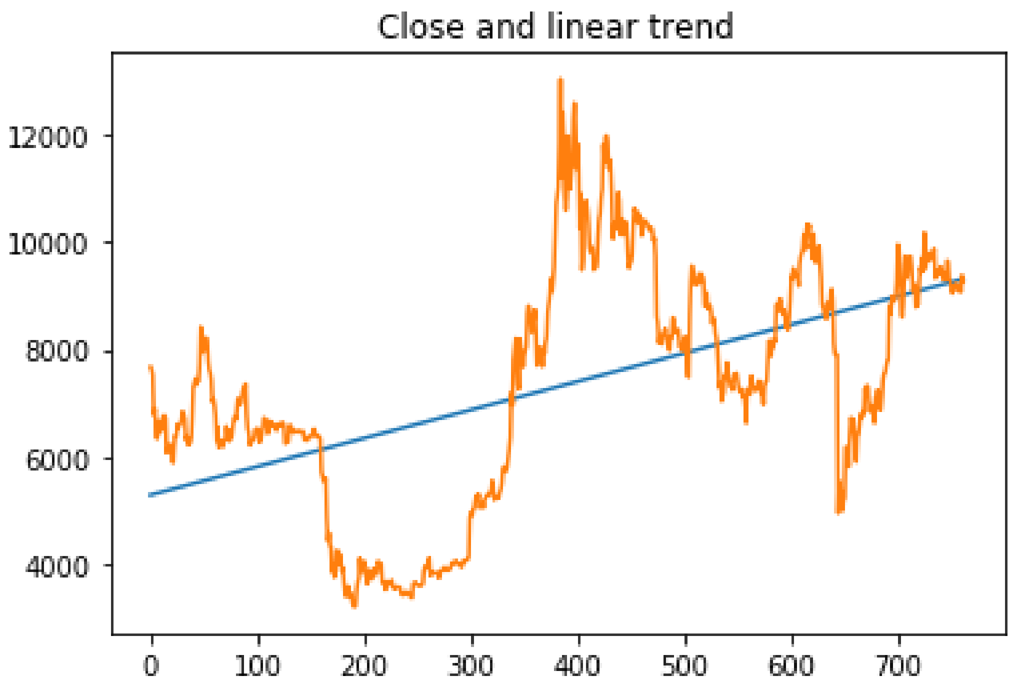
[6]:
```
create_laggedFeatures(df, 'Close', 1).head()
```

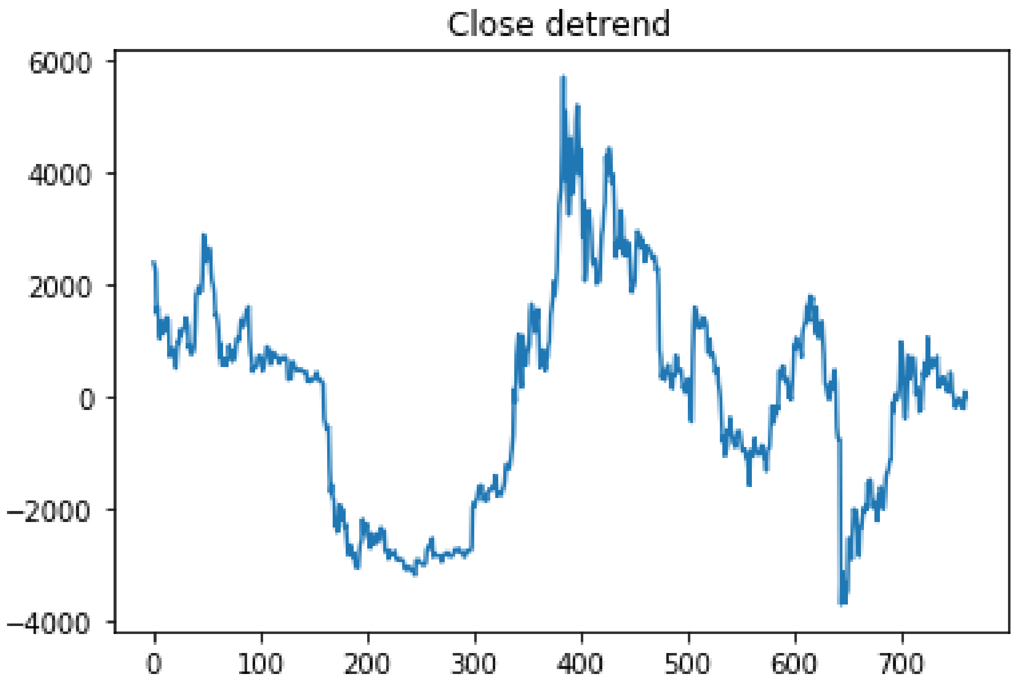
[6]:
```
         Date     Open     High      Low    Close      Volume  Dividends  \
0  2018-06-07  7650.82  7741.27  7650.82  7678.24  4485799936          0
1  2018-06-08  7685.14  7698.19  7558.40  7624.92  4227579904          0
2  2018-06-09  7632.52  7683.58  7531.98  7531.98  3845220096          0
3  2018-06-10  7499.55  7499.55  6709.07  6786.02  5804839936          0
4  2018-06-11  6799.29  6910.18  6706.63  6906.92  4745269760          0

   Stock Splits  day  month  year  Close_lagged
0             0    7      6  2018           NaN
1             0    8      6  2018       7678.24
2             0    9      6  2018       7624.92
3             0   10      6  2018       7531.98
4             0   11      6  2018       6786.02
```

[7]:
```
def detrend(dataframe, feature):
    """
    Input:
        dataframe(pandas DataFrame): BTC data
        feature(string): feature that wil be detrended
    Output:
        detrend(np array): detrended values of features
        plots
    """
    X = [i for i in range(0, len(df))]
    X = np.reshape(X, (len(X), 1))
    y = dataframe[feature].values
    model = LinearRegression()
```

```
    model.fit(X, y)
    trend = model.predict(X)
    plt.plot(X, trend)
    plt.plot(X, dataframe[feature].values)
    plt.title('{} and linear trend'.format(feature))
    plt.show()
    detrend = y - trend
    plt.plot(X, detrend)
    plt.title('{} detrend'.format(feature))
    plt.show()
    return detrend
```

[8]:
```
detrend = detrend(df, 'Close')
```



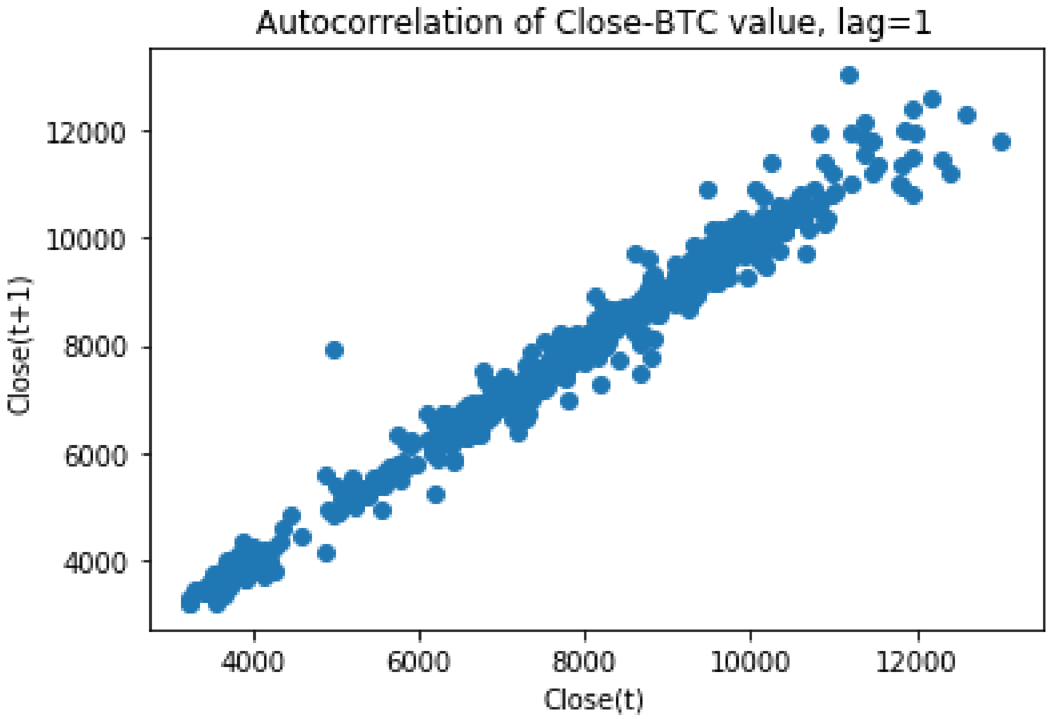


[9]:
```
detrend[-10:] # showing the last 10 values of detrended Close values
```

[9]:
```
array([-118.44072647,  -76.43103851, -134.55135056,  -49.4716626 ,
       -159.65197464, -201.02228668, -161.09259872, -224.90291076,
         71.3667772 ,  -36.35353484])
```

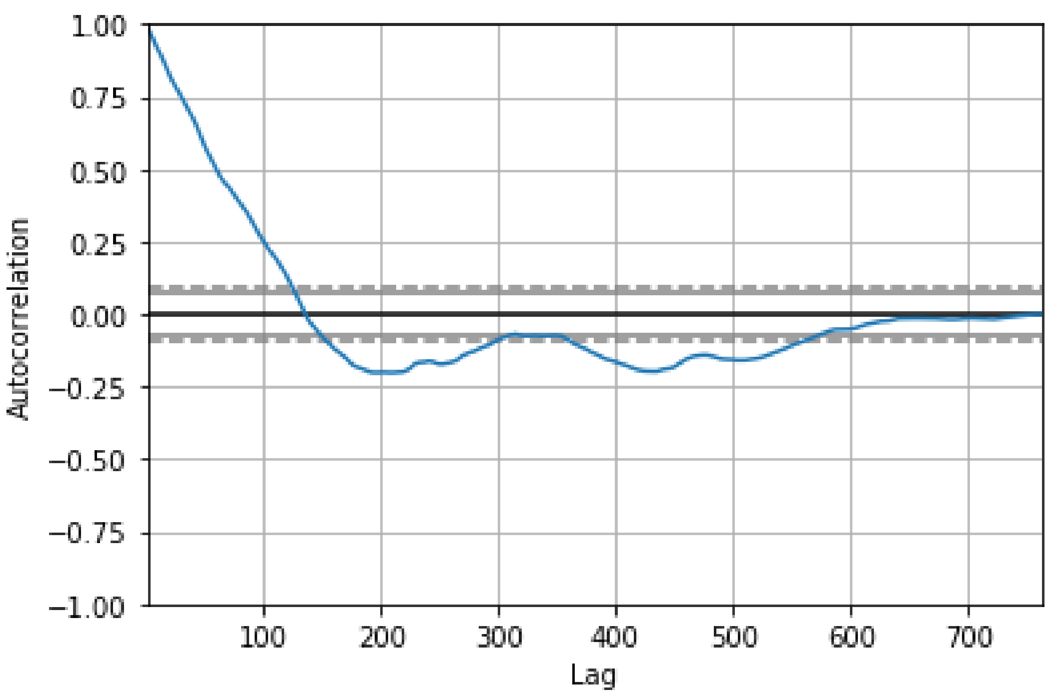
0.2 Autocorrelation visualization

[10]:
```
create_laggedFeatures(df, 'Close', 1)
plt.title('Autocorrelation of Close-BTC value, lag=1')
plt.scatter(df['Close'][1:], df['Close_lagged'][1:])
plt.xlabel('Close(t)')
plt.ylabel('Close(t+1)')
plt.show()
```



[11]:
```
autocorrelation_plot(df['Close'])
```

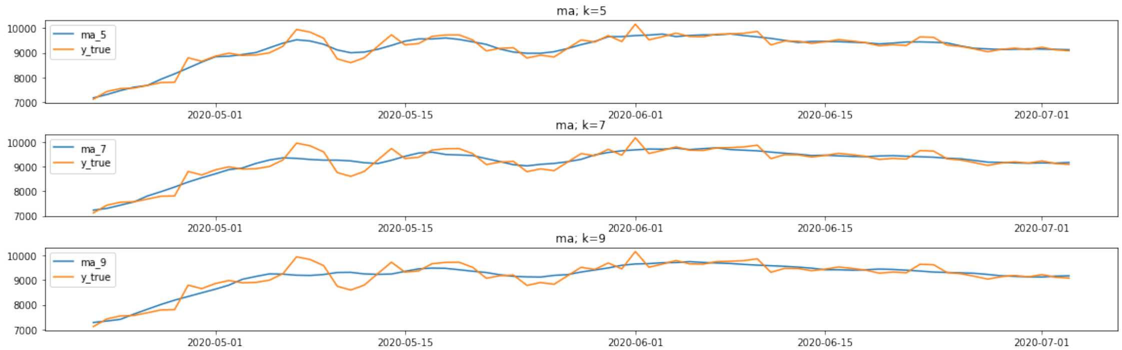
[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7d48e6ba20>



0.3 Moving average smoothing

[12]:
```
def ma_smoothing(dataframe, k):
    """
    Input:
        dataframe(pandas DataFrame): BTC data
        k(int): range of moving average
    Output:
        dataframe(pandas DataFrame): dataframe with the ma value
    """
    index = 0
    k_ = int((k-1)/2)
    ma_smoothing = []
    while index < len(dataframe)-k_:
        if index < k_:
            ma_smoothing.append(np.nan)
        else:
            ma_smoothing.append(dataframe[index-k_:index+k_+1]['Close'].mean())
        index += 1
    for i in range(k_):
        ma_smoothing.append(np.nan)
    dataframe['ma_'+str(k)] = ma_smoothing
    return dataframe

ma_smoothing(df, 5)
ma_smoothing(df, 7)
ma_smoothing(df, 9)
df_ = df[int(len(df)*9/10):len(df)-4][['Date', 'Close', 'ma_5', 'ma_7', 'ma_9']]

dates = df_['Date']
y_true = df_['Close']
data = {'ma_5': df_['ma_5'],
       'ma_7': df_['ma_7'],
       'ma_9': df_['ma_9']}

fig, axs = plt.subplots(len(list(data.keys())), 1, figsize=[16, 5])
fig.tight_layout()
c = 0
for key in data.keys():
    axs[c].plot(dates, data[key], label=key)
    axs[c].plot(dates, y_true, label='y_true')
    axs[c].set_title('ma; k={}'.format(key.split('_')[1]))
    axs[c].legend()
    c += 1
```

```
plt.show()
```



0.4 Modelling

[13]:
```
def calculate_RMSE(y_pred, y_true):
    """
    Input:
        y_pred(np vector): predicted values
        y_true(np vector): true values
    Output:
        RMSE(float): value of Root Mean Square Error
    """
    n = len(y_pred)
    RMSE = (np.mean((y_pred - y_true)**2))**0.5
    return RMSE
```

[14]:
```
# Setting the test size of the data
TEST_SIZE = int(len(df)*(9/10))
```

0.4.1 Auto Regressive Model

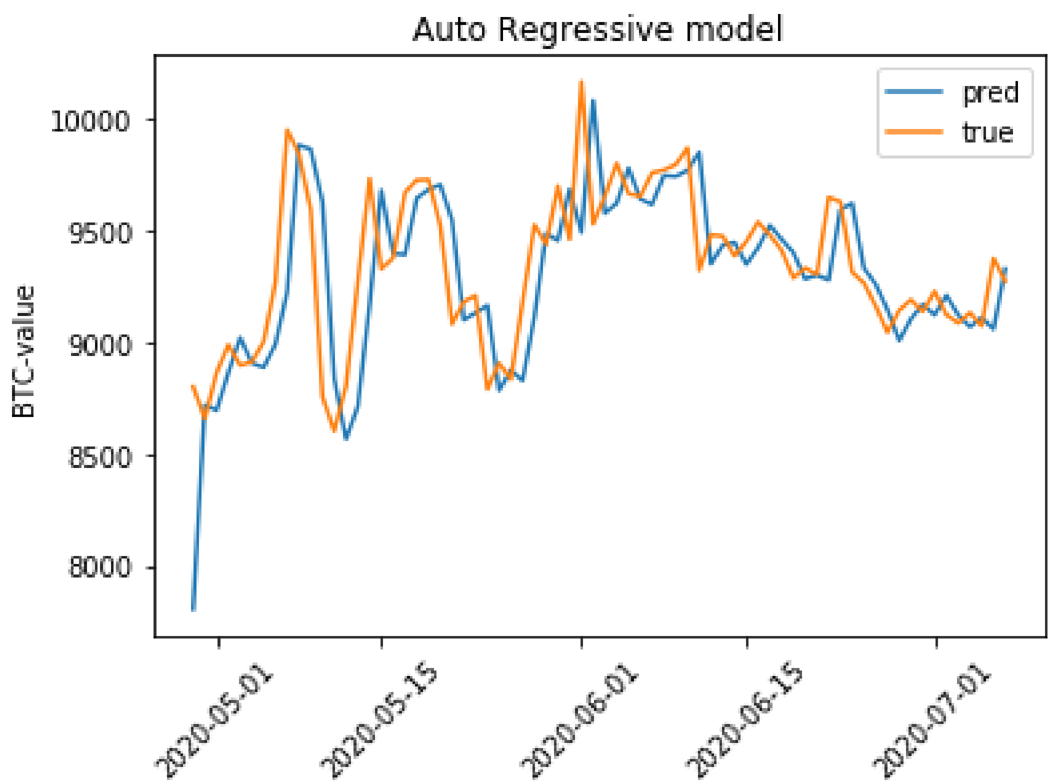
[15]:
```
avg_range = 7
index = len(df) - 1
X, y = [], []
while index-avg_range+1 > 0:
    X.append(df['Close'][index-avg_range:index].values)
    y.append(np.array([df.iloc[index]['Close']]))
    index = index - 1

X.reverse(), y.reverse()
X, y = np.array(X), np.array(y)
```

[16]:
```
X_train, y_train = X[:TEST_SIZE], y[:TEST_SIZE]
X_test, y_test = X[TEST_SIZE:], y[TEST_SIZE:]
```

```
model_ar = LinearRegression()
model_ar.fit(X_train, y_train)
```

[16]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

[17]:
```
y_pred = model_ar.predict(X_test)
y_true = y_test
dates = df['Date'][TEST_SIZE:][7:]
plt.plot(dates, y_pred, label='pred')
plt.plot(dates, y_true, label='true')
plt.xticks(rotation=45)
plt.title('Auto Regressive model')
plt.ylabel('BTC-value')
plt.legend()
plt.show()
```



[18]:
```
rmse_ar = calculate_RMSE(y_pred, y_true)
print('RMSE of Auto Regressive: {}'.format(round(rmse_ar, 2)))
```

RMSE of Auto Regressive: 290.16

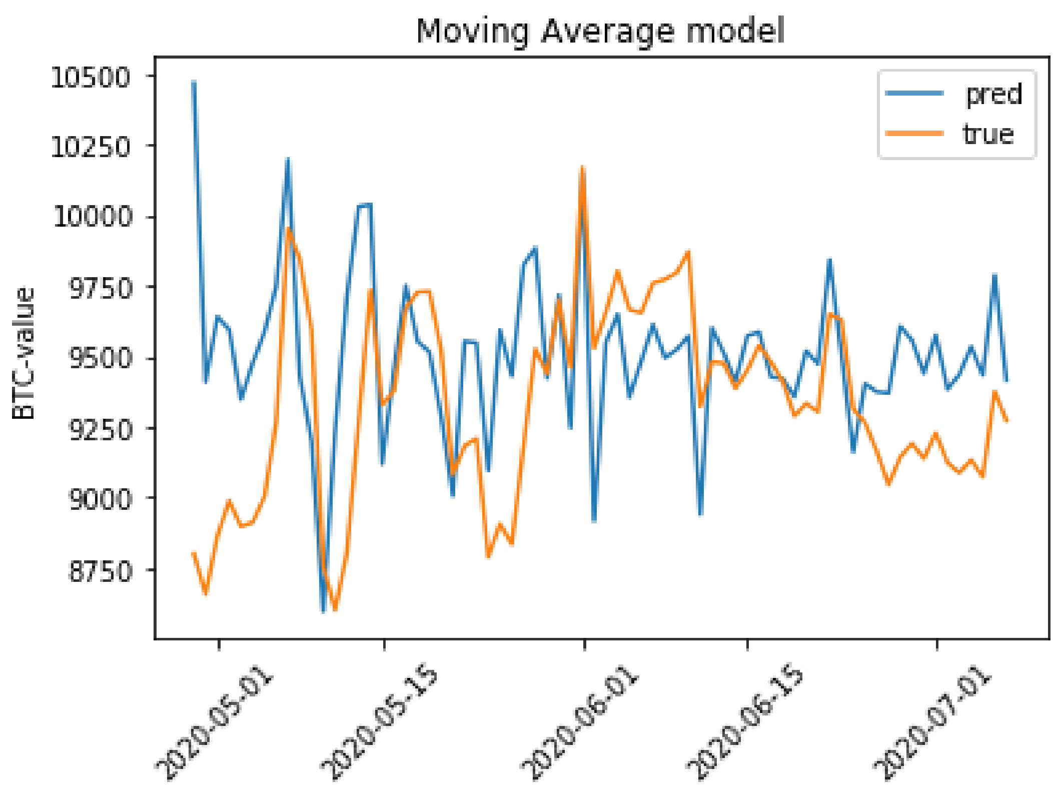
0.4.2 Moving Average Model

[179]:
```
error = model_ar.predict(X) - y
```

[180]:
```
X_train, y_train = error[:TEST_SIZE], y[:TEST_SIZE]
X_test, y_test = error[TEST_SIZE:], y[TEST_SIZE:]

model_ma = LinearRegression()
model_ma.fit(X_train, y_train)
```

[180]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

[185]:
```
y_pred = model_ma.predict(X_test)
c = np.mean(y_train)/3 # c constant
y_pred = y_pred + c
y_true = y_test
dates = df['Date'][TEST_SIZE:][7:]
plt.plot(dates, y_pred, label='pred')
plt.plot(dates, y_true, label='true')
plt.xticks(rotation=45)
plt.title('Moving Average model')
plt.ylabel('BTC-value')
plt.legend()
plt.show()
```



[22]:
```python
rmse_ma = calculate_RMSE(y_pred, y_true)
print('RMSE of Auto Regressive: {}'.format(round(rmse_ma, 2)))
```

RMSE of Auto Regressive: 379.51

0.4.3 LSTM Model

This is an example of the LSTM that I developed to predict stock values. This model uses the last 7 days values of the stock value do predict the current day. You can check the details of that network on train_model.py

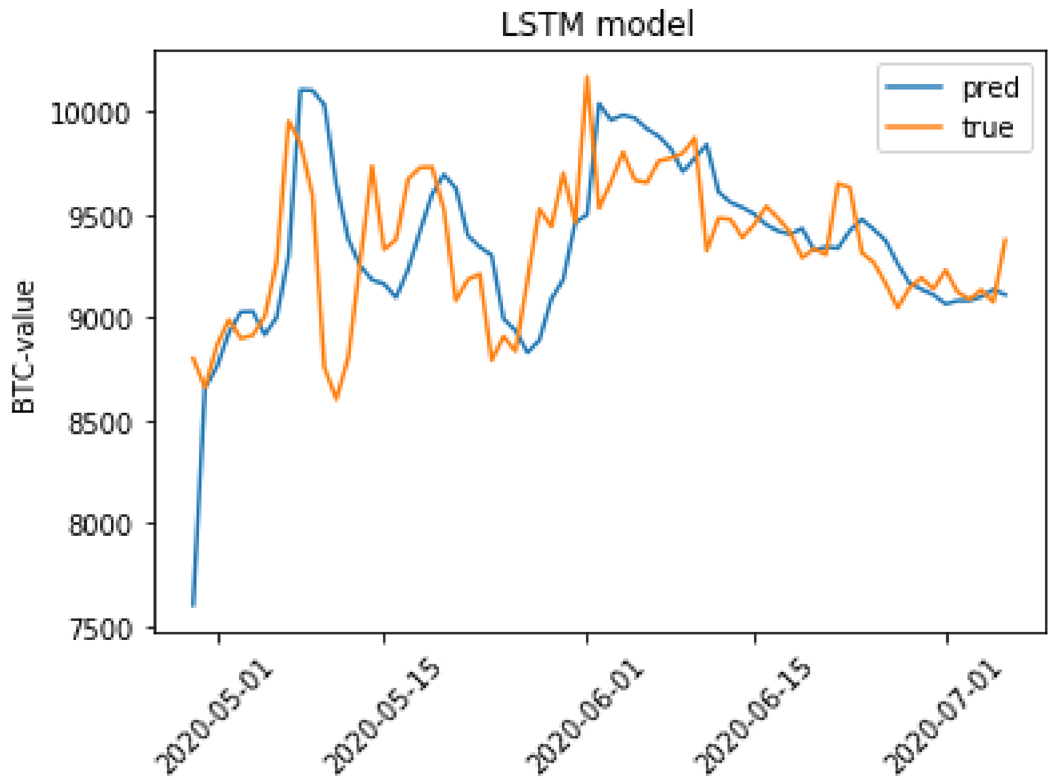
[75]:
```python
df['Date'] = pd.to_datetime(df['Date']).apply(lambda x: x.date())
df_train = df[:TEST_SIZE]
df_test = df[TEST_SIZE:]
```

[76]:
```python
model_lstm = trainModel(df_train)
```

```
Epoch 1 completed!
Epoch 2 completed!
Epoch 3 completed!
Epoch 4 completed!
Epoch 5 completed!
```

[166]:
```python
index = 0
y_pred = []
dates = []
while index+8 < len(df_test):
    df_test_ = df_test[index:index+8]
    vec = getFeatures(df_test_, len(df_test_))[:-1]
    z = standardScaler(vec)
    x = np.array([z[0]])
    x = x.reshape(x.shape[0], 1, x.shape[1])
    yPred = model_lstm.predict(x)
    minValue, maxValue = z[1][0], z[1][1]
    yPred = yPred*(maxValue - minValue) + minValue
    y_pred.append(yPred[0][0])
    dates.append(df_test_['Date'][-1:].values)
    index += 1

y_true = df_test[7:len(df_test)-1]['Close'].values
```

[178]:
```python
plt.plot(dates, y_pred, label='pred')
plt.plot(dates, y_true, label='true')
plt.xticks(rotation=45)
plt.title('LSTM model')
plt.ylabel('BTC-value')
plt.legend()
plt.show()
```



0.5 References

Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2. Accessed on <07/07/2020>.