4_model_building_LSTM_F13

November 18, 2022

0.1 4_model_building_LSTM_F13

Group , November 18, 2022 1. Eduardo Garcia 2. Nari Kim 3. Thi Anh Ba Dang 4. Vishnu Prabhakar 5. VS Chaitanya Madduri 6. Yumeng Zhang

Description: Applying the LSTM model on the daily data.

- We are builing a basic model using the columns and the target provided in the intial dataset.
- Used Feature Set 13 which are basic features and we have calculated the Lag and rolled data.

0.1.1 Pre requisites:

1. And add the shortcut of the drive link : https://drive.google.com/drive/folders/1F8P3UlqSE6lFpHyBidVArd to your personal drive.

Files: crypto_data_hour_cleaned_v2.csv - Hourly Data

0.1.2 Output files:

Files:LSTM_tw28.h5

[ ]:
```
# install packages
! pip install keras-tuner --upgrade
! pip install ta
! pip install seglearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: tensorflow-addons in
/usr/local/lib/python3.7/dist-packages (0.18.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-
packages (from tensorflow-addons) (21.3)
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-
packages (from tensorflow-addons) (2.7.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging->tensorflow-addons)
(3.0.9)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: keras-tuner in /usr/local/lib/python3.7/dist-
packages (1.1.3)
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from keras-tuner) (2.23.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from keras-tuner) (1.21.6)
Requirement already satisfied: tensorboard in /usr/local/lib/python3.7/dist-
packages (from keras-tuner) (2.9.1)
Requirement already satisfied: ipython in /usr/local/lib/python3.7/dist-packages
(from keras-tuner) (7.9.0)
Requirement already satisfied: kt-legacy in /usr/local/lib/python3.7/dist-
packages (from keras-tuner) (1.0.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-
packages (from keras-tuner) (21.3)
Requirement already satisfied: prompt-toolkit<2.1.0,>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from ipython->keras-tuner) (2.0.10)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-
packages (from ipython->keras-tuner) (0.7.5)
Requirement already satisfied: jedi>=0.10 in /usr/local/lib/python3.7/dist-
packages (from ipython->keras-tuner) (0.18.1)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-
packages (from ipython->keras-tuner) (5.1.1)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-
packages (from ipython->keras-tuner) (2.6.1)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.7/dist-packages (from ipython->keras-tuner) (57.4.0)
Requirement already satisfied: pexpect in /usr/local/lib/python3.7/dist-packages
(from ipython->keras-tuner) (4.8.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from ipython->keras-tuner) (4.4.2)
Requirement already satisfied: backcall in /usr/local/lib/python3.7/dist-
packages (from ipython->keras-tuner) (0.2.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
/usr/local/lib/python3.7/dist-packages (from jedi>=0.10->ipython->keras-tuner)
(0.8.3)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages
(from prompt-toolkit<2.1.0,>=2.0.0->ipython->keras-tuner) (0.2.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-
packages (from prompt-toolkit<2.1.0,>=2.0.0->ipython->keras-tuner) (1.15.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging->keras-tuner) (3.0.9)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-
packages (from pexpect->ipython->keras-tuner) (0.7.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->keras-tuner) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->keras-tuner) (2022.9.24)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->keras-tuner) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in
```

```
/usr/local/lib/python3.7/dist-packages (from requests->keras-tuner) (3.0.4)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in
/usr/local/lib/python3.7/dist-packages (from tensorboard->keras-tuner) (3.19.6)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in
/usr/local/lib/python3.7/dist-packages (from tensorboard->keras-tuner) (0.6.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-
packages (from tensorboard->keras-tuner) (3.4.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/usr/local/lib/python3.7/dist-packages (from tensorboard->keras-tuner) (0.4.6)
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.7/dist-
packages (from tensorboard->keras-tuner) (1.50.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.7/dist-packages (from tensorboard->keras-tuner) (2.14.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/usr/local/lib/python3.7/dist-packages (from tensorboard->keras-tuner) (1.8.1)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.7/dist-
packages (from tensorboard->keras-tuner) (1.0.1)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.7/dist-
packages (from tensorboard->keras-tuner) (0.38.3)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.7/dist-
packages (from tensorboard->keras-tuner) (1.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-
packages (from google-auth<3,>=1.6.3->tensorboard->keras-tuner) (4.9)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from google-
auth<3,>=1.6.3->tensorboard->keras-tuner) (5.2.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.7/dist-packages (from google-
auth<3,>=1.6.3->tensorboard->keras-tuner) (0.2.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.7/dist-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tensorboard->keras-tuner) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in
/usr/local/lib/python3.7/dist-packages (from
markdown>=2.6.8->tensorboard->keras-tuner) (4.13.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard->keras-
tuner) (3.10.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/usr/local/lib/python3.7/dist-packages (from importlib-
metadata>=4.4->markdown>=2.6.8->tensorboard->keras-tuner) (4.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard->keras-tuner) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-
packages (from requests-oauthlib>=0.7.0->google-auth-
oauthlib<0.5,>=0.4.1->tensorboard->keras-tuner) (3.2.2)
```

1 Load and transform data

[ ]:
```
# Connecting to the google drive
from google.colab import drive
drive.mount('/content/drive')
from IPython.display import clear_output
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.model_selection import RandomizedSearchCV

#picking models for prediction.
from sklearn.svm import SVC
```

[ ]:
```
# file path
folder_path = '/content/drive/MyDrive/MADS_23_DL_final_project'
daily = pd.read_csv(folder_path + '/data/crypto_data_daily_cleaned_v1.csv')
```

[ ]:
```
daily.head()
```

[ ]:
```
    Open Time     Open    High      Low    Close     Volume train_test Crypto
0  2013-04-01   93.155  105.90   93.155  104.750  11008.524      Train    BTC
1  2013-04-02  104.720  127.00   99.000  123.016  24187.398      Train    BTC
2  2013-04-03  123.001  146.88  101.511  125.500  31681.780      Train    BTC
3  2013-04-04  125.500  143.00  125.500  135.632  15035.206      Train    BTC
4  2013-04-05  136.000  145.00  135.119  142.990  11697.741      Train    BTC
```

[ ]:
```
daily
```

[ ]:
```
        Open Time     Open     High      Low    Close          Volume  \
0      2013-04-01   93.155  105.900   93.155  104.750    11008.524000
1      2013-04-02  104.720  127.000   99.000  123.016    24187.398000
2      2013-04-03  123.001  146.880  101.511  125.500    31681.780000
3      2013-04-04  125.500  143.000  125.500  135.632    15035.206000
4      2013-04-05  136.000  145.000  135.119  142.990    11697.741000
…             …        …        …        …        …               …
20741  2022-09-26   28.350   28.606   27.490   28.460   231982.933089
20742  2022-09-27   28.460   30.232   27.607   28.130   371967.089380
20743  2022-09-28   28.147   28.280   26.640   27.630   181253.911464
20744  2022-09-29   27.649   28.295   27.000   27.790   155764.477192
```

```
20745  2022-09-30   27.830   28.362   27.290   27.730  150287.772536

      train_test Crypto
0          Train    BTC
1          Train    BTC
2          Train    BTC
3          Train    BTC
4          Train    BTC
…          …    …
20741       Test    ETC
20742       Test    ETC
20743       Test    ETC
20744       Test    ETC
20745       Test    ETC

[20746 rows x 8 columns]
```

1.1 Train / Test Split

[ ]:
```
daily['train_test'].value_counts(normalize=True)
```

[ ]:
```
Train    0.82546
Test     0.17454
Name: train_test, dtype: float64
```

[ ]:
```
df = daily.copy()
```

[ ]:
```
# train test split
train_df = df[df['train_test']=='Train']
test_df = df[df['train_test']=='Test']
```

[ ]:
```
train_df.head()
```

[ ]:

	Open Time	Open	High	Low	Close	Volume	train_test	Crypto
0	2013-04-01	93.155	105.90	93.155	104.750	11008.524	Train	BTC
1	2013-04-02	104.720	127.00	99.000	123.016	24187.398	Train	BTC
2	2013-04-03	123.001	146.88	101.511	125.500	31681.780	Train	BTC
3	2013-04-04	125.500	143.00	125.500	135.632	15035.206	Train	BTC
4	2013-04-05	136.000	145.00	135.119	142.990	11697.741	Train	BTC

[ ]:
```
train_df.columns
```

[ ]:
```
Index(['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'train_test',
       'Crypto'],
      dtype='object')
```

1.2 Calculate percentage change for Train and Test

[ ]:
```
def calculate_pct_change(df):
  coins = df.Crypto.unique()
  df_pct_change = pd.DataFrame()
  for coin in coins:
    x = df[df['Crypto']==coin]
    x['pct_change_1day'] = x['Close'].pct_change(1)
    df_pct_change = pd.concat([df_pct_change,x])
  return df_pct_change
```

[ ]:
```
train_df = calculate_pct_change(train_df)
test_df = calculate_pct_change(test_df)
```

[ ]:
```
test_df.head()
```

[ ]:
```
       Open Time      Open      High       Low     Close     Volume  \
3098  2021-10-01  43828.89  48500.00  43287.44  48165.76  38375.517
3099  2021-10-02  48185.61  48361.83  47438.00  47657.69  12310.011
3100  2021-10-03  47649.00  49300.00  47119.87  48233.99  14411.104
3101  2021-10-04  48233.99  49530.53  46895.80  49245.54  25695.213
3102  2021-10-05  49244.13  51922.00  49057.18  51493.99  30764.491

     train_test Crypto  pct_change_1day
3098       Test    BTC              NaN
3099       Test    BTC        -0.010548
3100       Test    BTC         0.012092
3101       Test    BTC         0.020972
3102       Test    BTC         0.045658
```

1.3 Generate lag features

[ ]:
```
# function to calculate the market capitalization of coins at each time point
def create_market_volumn_features(df):

  # calculate value of each cryto at certain time points
  df['Total_Value'] = df['Close']*df['Volume']
  # the sum of values at each time point
  sum_at_timepoints = df.groupby('Open Time').sum()['Total_Value']
  merged = df.merge(sum_at_timepoints, how='left',
                            on='Open Time', suffixes=('','_market'))
  merged['Value_Weight'] = merged['Total_Value']/merged['Total_Value_market']

  return merged
```

[ ]:
```
# function to create 1 day lag
def create_shift_features(df, col = 'pct_change_1day'):
   df['1_d_lag'] = df[col].shift(periods=1)
```

```
#  df['1_w_lag'] = df[col].shift(periods=7)
 return df
```

[ ]:
```
# stack all functions together and iterate through all coins
from ta import add_all_ta_features
def create_analysis_colums(df):

  master_df = pd.DataFrame()
  crypto_coins = df['Crypto'].unique()

  for coin in crypto_coins:

    temp_df = df[df['Crypto']==coin]
    temp_df['pct_change_1day'] = temp_df['Close'].pct_change()
    temp_df = create_shift_features(temp_df.copy(),col = 'pct_change_1day')
    temp_df = add_all_ta_features(temp_df.copy(), open="Open", high="High",␣
 ↪low="Low", close="Close", volume="Volume", fillna=True)
    if master_df.empty :
      master_df = temp_df
    else:
      master_df = pd.concat([master_df, temp_df])
  return master_df
```

[ ]:
```
# apply all functions
train_df =create_analysis_colums(train_df)
test_df =create_analysis_colums(test_df)
```

[ ]:
```
# calculate the market capitalization
train_df =create_market_volumn_features(train_df.copy())
test_df =create_market_volumn_features(test_df.copy())
```

[ ]:
```
# features selected from previous model importance analysis
feat_to_keep = ['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume',␣
 ↪'train_test',
       'Crypto',␣
 ↪'pct_change_1day','volatility_kcw','trend_cci','volume_adi','momentum_ppo_hist','momentum_s
             'volatility_dcw','volume_vpt','volatility_bbw','Total_Value',␣
 ↪'Total_Value_market', 'Value_Weight']

lag_cols = [col for col in train_df.columns if 'lag' in col]
feat_to_keep.extend(lag_cols)
```

[ ]:
```
# only keep above features
train_df = train_df[feat_to_keep]
test_df = test_df[feat_to_keep]
```

```
[ ]: # function to shift all technical features by 1 because it causes leakage
def shift_vol(df):
  impo_feat =␣
↪['volatility_kcw','trend_cci','volume_adi','momentum_ppo_hist','momentum_stoch','volatility
               'volatility_dcw','volume_vpt','volatility_bbw']
  master_df = pd.DataFrame()
  crypto_coins = df['Crypto'].unique()


  for coin in crypto_coins:
    temp_df = df[df['Crypto']==coin]
    for feat in impo_feat:

      temp_df[feat] = temp_df[feat].shift(1)
    if master_df.empty :
      master_df = temp_df
    else:
      master_df = pd.concat([master_df, temp_df])
  return master_df
```

```
[ ]: # apply shift function
train_df = shift_vol(train_df.copy())
test_df = shift_vol(test_df.copy())
```

```
[ ]:
```

1.4 Extract year, month, day, hour and weekday from time stamp

1.4.1 Encoding of ordinals

```
[ ]: def encode_cyclicals(df_x):

    df_x['month_sin'] = np.sin(2*np.pi*df_x.month/12)
    df_x['month_cos'] = np.cos(2*np.pi*df_x.month/12)
    df_x.drop('month', axis=1, inplace=True)

    df_x['day_sin'] = np.sin(2*np.pi*df_x.day/31)
    df_x['day_cos'] = np.cos(2*np.pi*df_x.day/31)
    df_x.drop('day', axis=1, inplace=True)

    df_x['dayofweek_sin'] = np.sin(2*np.pi*df_x.weekday/7)
    df_x['dayofweek_cos'] = np.cos(2*np.pi*df_x.weekday/7)
    df_x.drop('weekday', axis=1, inplace=True)

    df_x['hour_sin'] = np.sin(2*np.pi*df_x.hour/24)
    df_x['hour_cos'] = np.cos(2*np.pi*df_x.hour/24)
    df_x.drop('hour', axis=1, inplace=True)
```

```
    df_x['hour_sin'] = np.sin(2*np.pi*df_x.minute/60)
    df_x['hour_cos'] = np.cos(2*np.pi*df_x.minute/60)
    df_x.drop('minute', axis=1, inplace=True)


    return df_x
```

[ ]:
```
# function to extract time features and encode them
def date_values_extraction(new_df):
  df = new_df.copy()
  df['year'] = pd.DatetimeIndex(df['Open Time']).year
  df['month'] = pd.DatetimeIndex(df['Open Time']).month
  df['day'] = pd.DatetimeIndex(df['Open Time']).day
  df['weekday'] = pd.DatetimeIndex(df['Open Time']).dayofweek


  df['Open Time'] = pd.to_datetime(df['Open Time'])
  df['minute'] = df['Open Time'].dt.minute
  df['hour'] = df['Open Time'].dt.hour
  df = encode_cyclicals(df.copy())
  return df
```

[ ]:
```
# apply above function
train_df = date_values_extraction(train_df)
test_df = date_values_extraction(test_df)
```

1.5 One hot coding the coins

[ ]:
```
# Applying one hot encoding on Crypto Coin
def crypto_one_hot_encoding(df):
  y_dummies = pd.get_dummies(df['Crypto'], prefix='Crypto', drop_first= False)
  # creating a additional column if the model is used for new coin.
  y_dummies['other_crypto'] =0
  return y_dummies
```

[ ]:
```
# roll data for one hot encoded dummy variables
def rolling_hot_data(train_df,TIME_WINDOW):
  train_hot = pd.DataFrame()
  for col in train_df.columns:
    if train_hot.empty:
        train_hot = train_df[train_df[col]==1][:-TIME_WINDOW]
    else:
      train_hot = pd.concat([train_hot, train_df[train_df[col]==1][:
 ↪-TIME_WINDOW]], axis=0)
  return train_hot
```

```
[ ]: # one hot encoding the coins
train_onehot_data = crypto_one_hot_encoding(train_df)
test_onehot_data = crypto_one_hot_encoding(test_df)
```

```
[ ]: # roll the one hot encoded data
TIME_WINDOW=28
train_onehot_data = rolling_hot_data(train_onehot_data,TIME_WINDOW)
test_onehot_data = rolling_hot_data(test_onehot_data,TIME_WINDOW)
```

```
[ ]: test_onehot_data.shape
```

```
[ ]: (3341, 11)
```

```
[ ]: train_df['pct_change_1day'].describe()
```

```
[ ]: count    17115.000000
     mean         0.005533
     std          0.162083
     min         -0.564847
     25%         -0.025641
     50%          0.000000
     75%          0.028824
     max         19.058824
     Name: pct_change_1day, dtype: float64
```

```
[ ]: test_df['pct_change_1day'].describe()
```

```
[ ]: count    3611.000000
     mean       -0.001452
     std         0.046840
     min        -0.204696
     25%        -0.026274
     50%         0.000000
     75%         0.024091
     max         0.344702
     Name: pct_change_1day, dtype: float64
```

2 Defining the Target Variable

We devide the data into 3 classes, one is with return below 0, one is above 0 but below market rate of return, one is above market rate of return. The market rate of return we use here is annual return of S&P 500 index in 2021.

```
[ ]: # function to create classes
def create_target(df, prob='Classification'):
  if prob == 'Classification':
    market_RoR = 26.89
```

```python
        market_RoR_1d = market_RoR/365
        df['Target'] = np.where(df['pct_change_1day']>0, 1,0)
        df['Target'] = np.where(df['pct_change_1day']>market_RoR_1d, 2,1)
        df['Target'][df['Target']==1] = np.
    ↪where(df['pct_change_1day'][df['Target']==1]>=0, 1,0)
    elif prob == 'Regression':
        df['Target'] = df['pct_change_1day']
    return df
```

[ ]:
```python
# apply function
train_df = create_target(train_df, 'Classification')
test_df = create_target(test_df, 'Classification')
```

[ ]:
```python
train_df['Target'].value_counts(normalize=True)
```

[ ]:
```
0    0.460088
1    0.449343
2    0.090569
Name: Target, dtype: float64
```

[ ]:
```python
test_df['Target'].value_counts(normalize=True)
```

[ ]:
```
0    0.491025
1    0.464513
2    0.044463
Name: Target, dtype: float64
```

[ ]:
```python
test_df.drop(['pct_change_1day'], axis=1, inplace=True) # droppping the column␣
 ↪as we already extracted the target
train_df.drop(['pct_change_1day'], axis=1, inplace=True) # droppping the column␣
 ↪as we already extracted the target
```

[ ]:
```python
train_df.shape
```

[ ]:
```
(17125, 32)
```

[ ]:
```python
test_df.shape
```

[ ]:
```
(3621, 32)
```

[ ]:
```python
# define the targets
target = train_df['Target']
test_target = test_df['Target']
```

[ ]:
```python

```

3 Drop columns

[ ]:
```
# drop unuseful columns
train_df.drop(['Target','Open Time','train_test',],axis=1,inplace=True)
test_df.drop(['Target','Open Time','train_test',],axis=1,inplace=True)
```

[ ]:
```
# dropping the list of the columns

drop_columns = ['Open','Close']



if drop_columns:
  norm_train_df = train_df.drop(drop_columns,axis=1)
  norm_test_df = test_df.drop(drop_columns,axis=1)
else:
  norm_train_df = train_df
  norm_test_df = test_df
```

[ ]:
```
norm_train_df.columns
```

[ ]:
```
Index(['High', 'Low', 'Volume', 'Crypto', 'volatility_kcw', 'trend_cci',
       'volume_adi', 'momentum_ppo_hist', 'momentum_stoch', 'volatility_kcp',
       'volume_em', 'volatility_dcw', 'volume_vpt', 'volatility_bbw',
       'Total_Value', 'Total_Value_market', 'Value_Weight', '1_d_lag', 'year',
       'month_sin', 'month_cos', 'day_sin', 'day_cos', 'dayofweek_sin',
       'dayofweek_cos', 'hour_sin', 'hour_cos'],
      dtype='object')
```

[ ]:
```
norm_train_df.shape
```

[ ]:
```
(17125, 27)
```

[ ]:
```
target.shape
```

[ ]:
```
(17125,)
```

[ ]:
```
norm_train_df.head()
```

[ ]:
```
     High      Low     Volume Crypto  volatility_kcw  trend_cci     volume_adi  \
0  105.90   93.155  11008.524    BTC             NaN        NaN            NaN
1  127.00   99.000  24187.398    BTC       25.170751   0.000000    9021.893541
2  146.88  101.511  31681.780    BTC       37.448244  66.666667   26326.249139
3  143.00  125.500  15035.206    BTC       50.324142  82.362074   28148.148762
4  145.00  135.119  11697.741    BTC       43.448762  98.871690   30522.852156

   momentum_ppo_hist  momentum_stoch  volatility_kcp  …  1_d_lag  year  \
0                NaN             NaN             NaN  …      NaN  2013
1           0.000000       90.976854        0.636590  …      NaN  2013
```

```
2           1.098643       88.228690        0.848816  …  0.174377  2013
3           1.848551       60.204746        0.698938  …  0.020192  2013
4           2.762857       79.063751        0.816463  …  0.080733  2013

   month_sin  month_cos   day_sin   day_cos  dayofweek_sin  dayofweek_cos  \
0   0.866025       -0.5  0.201299  0.979530       0.000000       1.000000
1   0.866025       -0.5  0.394356  0.918958       0.781831       0.623490
2   0.866025       -0.5  0.571268  0.820763       0.974928      -0.222521
3   0.866025       -0.5  0.724793  0.688967       0.433884      -0.900969
4   0.866025       -0.5  0.848644  0.528964      -0.433884      -0.900969

   hour_sin  hour_cos
0       0.0       1.0
1       0.0       1.0
2       0.0       1.0
3       0.0       1.0
4       0.0       1.0

[5 rows x 27 columns]
```

[ ]:
```
norm_test_df.head()
```

[ ]:
```
       High       Low     Volume Crypto  volatility_kcw  trend_cci  \
0  48500.00  43287.44  38375.517    BTC             NaN        NaN
1  48361.83  47438.00  12310.011    BTC       22.347013   0.000000
2  49300.00  47119.87  14411.104    BTC       12.991160  66.666667
3  49530.53  46895.80  25695.213    BTC       11.656914  71.870986
4  51922.00  49057.18  30764.491    BTC       11.452558  85.715360

     volume_adi  momentum_ppo_hist  momentum_stoch  volatility_kcp  …  \
0           NaN                NaN             NaN             NaN  …
1  33454.083845           0.000000       93.587796        0.645293  …
2  26998.800050          -0.067370       83.840762        0.568863  …
3  27316.816963          -0.029487       82.270281        0.621071  …
4  47453.298136           0.129936       95.435113        0.761919  …

    1_d_lag  year  month_sin  month_cos   day_sin   day_cos  dayofweek_sin  \
0       NaN  2021  -0.866025        0.5  0.201299  0.979530      -0.433884
1       NaN  2021  -0.866025        0.5  0.394356  0.918958      -0.974928
2 -0.010548  2021  -0.866025        0.5  0.571268  0.820763      -0.781831
3  0.012092  2021  -0.866025        0.5  0.724793  0.688967       0.000000
4  0.020972  2021  -0.866025        0.5  0.848644  0.528964       0.781831

   dayofweek_cos  hour_sin  hour_cos
0      -0.900969       0.0       1.0
1      -0.222521       0.0       1.0
2       0.623490       0.0       1.0
```

```
3       1.000000       0.0       1.0
4       0.623490       0.0       1.0

[5 rows x 27 columns]
```

[ ]:

```

```

4 Roll the train and test data

[ ]:

```
# use past 28 days to forecast the next day
TIME_WINDOW=28
FORECAST_DISTANCE=1
```

[ ]:

```
from seglearn.transform import FeatureRep, SegmentXYForecast, last

# function to roll data
def Segment_multi(train_df, target):
  master_df_x = pd.DataFrame()
  master_df_y = pd.DataFrame()

  crypto_coins = df['Crypto'].unique()

  segmenter = SegmentXYForecast(width=TIME_WINDOW, step=1, y_func=last,␣
↪forecast=FORECAST_DISTANCE)
  for coin in crypto_coins:

    coin_index = train_df[train_df['Crypto']==coin].index
    X_train_rolled, y_train_rolled,_=segmenter.fit_transform([train_df.
↪iloc[coin_index].drop(['Crypto'], axis=1).values],[target.iloc[coin_index].
↪values])


    if coin == 'BTC' :
      master_df_x = X_train_rolled
      master_df_y = y_train_rolled
    else:
      master_df_x = np.concatenate([master_df_x, X_train_rolled], axis=0)

      master_df_y = np.concatenate([master_df_y, y_train_rolled], axis=0)

  return master_df_x, master_df_y
```

[ ]:

```
# roll train and test data
train_roll, y_roll =  Segment_multi(norm_train_df.fillna(0), target)
test_roll, y_test_roll =  Segment_multi(norm_test_df.fillna(0), test_target)
```

[ ]:
```
norm_train_df.columns
```

[ ]:
```
Index(['High', 'Low', 'Volume', 'Crypto', 'volatility_kcw', 'trend_cci',
       'volume_adi', 'momentum_ppo_hist', 'momentum_stoch', 'volatility_kcp',
       'volume_em', 'volatility_dcw', 'volume_vpt', 'volatility_bbw',
       'Total_Value', 'Total_Value_market', 'Value_Weight', '1_d_lag', 'year',
       'month_sin', 'month_cos', 'day_sin', 'day_cos', 'dayofweek_sin',
       'dayofweek_cos', 'hour_sin', 'hour_cos'],
      dtype='object')
```

[ ]:
```
train_roll.shape
```

[ ]:
```
(16845, 28, 26)
```

[ ]:
```
test_roll.shape
```

[ ]:
```
(3341, 28, 26)
```

5 Evaluation function

[ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
def generate_model_report(y_actual, y_predicted, metric_type):
    print("=================Printing the {} metrics==================".
 ↪format(metric_type))
    if metric_type=='micro':
      print("Accuracy = " , round(accuracy_score(y_actual, y_predicted),4))
    print("Precision = " ,round(precision_score(y_actual, y_predicted,␣
 ↪average=metric_type),4))
    print("Recall = " ,round(recall_score(y_actual, y_predicted,␣
 ↪average=metric_type),4))
    print("F1 Score = " ,round(f1_score(y_actual, y_predicted,␣
 ↪average=metric_type),4))
    print("=====================================")
```

[ ]:
```

```

6 stateless LSTM model

[ ]:
```
from tensorflow import keras
from kerastuner.tuners import RandomSearch
from tensorflow.keras.layers import Dense, Dropout, LSTM, Bidirectional,␣
 ↪BatchNormalization, Input, concatenate
from tensorflow.keras.models import Model
from tensorflow.keras import backend as be
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

```python
import keras_tuner
```

[ ]:
```python
# use 2 inputs because LSTM cannot process non-sequential data
# so we use another dense layer to handle coin dummy data
float_input = Input(shape=(train_roll.shape[1],train_roll.shape[2]))
one_hot_input = Input(shape=(11,) )

first_lstm = LSTM(60)(float_input)#,stateful=True
# we also tried stateful/bidirectional LSTM but no improvement on the result
first_dense = Dense(50)(one_hot_input)
merge_one = concatenate([first_dense, first_lstm])
dense_inner = Dense(10)(merge_one)
dense_output = Dense(3, activation='softmax')(dense_inner)

model = Model(inputs=[float_input, one_hot_input], outputs=dense_output)
model.compile(loss='sparse_categorical_crossentropy', # we select it because␣
 ↪it's multi-class classification
              optimizer='Adam',
              metrics=['sparse_categorical_crossentropy'])
#model.summary()
model.fit([train_roll,train_onehot_data], y_roll, epochs=20,
          validation_data = ([test_roll,test_onehot_data],␣
 ↪y_test_roll))#batch_size=BATCH_SIZE,
```

```
Epoch 1/20
527/527 [==============================] - 12s 18ms/step - loss: 0.9503 -
sparse_categorical_crossentropy: 0.9503 - val_loss: 0.8651 -
val_sparse_categorical_crossentropy: 0.8651
Epoch 2/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9288 -
sparse_categorical_crossentropy: 0.9288 - val_loss: 0.8682 -
val_sparse_categorical_crossentropy: 0.8682
Epoch 3/20
527/527 [==============================] - 10s 18ms/step - loss: 0.9257 -
sparse_categorical_crossentropy: 0.9257 - val_loss: 0.8562 -
val_sparse_categorical_crossentropy: 0.8562
Epoch 4/20
527/527 [==============================] - 9s 18ms/step - loss: 0.9244 -
sparse_categorical_crossentropy: 0.9244 - val_loss: 0.8758 -
val_sparse_categorical_crossentropy: 0.8758
Epoch 5/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9234 -
sparse_categorical_crossentropy: 0.9234 - val_loss: 0.8802 -
val_sparse_categorical_crossentropy: 0.8802
Epoch 6/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9232 -
sparse_categorical_crossentropy: 0.9232 - val_loss: 0.8732 -
val_sparse_categorical_crossentropy: 0.8732
```

```
Epoch 7/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9224 -
sparse_categorical_crossentropy: 0.9224 - val_loss: 0.8827 -
val_sparse_categorical_crossentropy: 0.8827
Epoch 8/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9218 -
sparse_categorical_crossentropy: 0.9218 - val_loss: 0.8597 -
val_sparse_categorical_crossentropy: 0.8597
Epoch 9/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9217 -
sparse_categorical_crossentropy: 0.9217 - val_loss: 0.8733 -
val_sparse_categorical_crossentropy: 0.8733
Epoch 10/20
527/527 [==============================] - 9s 18ms/step - loss: 0.9216 -
sparse_categorical_crossentropy: 0.9216 - val_loss: 0.8774 -
val_sparse_categorical_crossentropy: 0.8774
Epoch 11/20
527/527 [==============================] - 9s 18ms/step - loss: 0.9209 -
sparse_categorical_crossentropy: 0.9209 - val_loss: 0.8615 -
val_sparse_categorical_crossentropy: 0.8615
Epoch 12/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9205 -
sparse_categorical_crossentropy: 0.9205 - val_loss: 0.8677 -
val_sparse_categorical_crossentropy: 0.8677
Epoch 13/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9208 -
sparse_categorical_crossentropy: 0.9208 - val_loss: 0.8802 -
val_sparse_categorical_crossentropy: 0.8802
Epoch 14/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9206 -
sparse_categorical_crossentropy: 0.9206 - val_loss: 0.8824 -
val_sparse_categorical_crossentropy: 0.8824
Epoch 15/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9205 -
sparse_categorical_crossentropy: 0.9205 - val_loss: 0.8775 -
val_sparse_categorical_crossentropy: 0.8775
Epoch 16/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9203 -
sparse_categorical_crossentropy: 0.9203 - val_loss: 0.8863 -
val_sparse_categorical_crossentropy: 0.8863
Epoch 17/20
527/527 [==============================] - 10s 18ms/step - loss: 0.9204 -
sparse_categorical_crossentropy: 0.9204 - val_loss: 0.8894 -
val_sparse_categorical_crossentropy: 0.8894
Epoch 18/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9207 -
sparse_categorical_crossentropy: 0.9207 - val_loss: 0.8650 -
val_sparse_categorical_crossentropy: 0.8650
```

```
Epoch 19/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9198 -
sparse_categorical_crossentropy: 0.9198 - val_loss: 0.8891 -
val_sparse_categorical_crossentropy: 0.8891
Epoch 20/20
527/527 [==============================] - 9s 17ms/step - loss: 0.9201 -
sparse_categorical_crossentropy: 0.9201 - val_loss: 0.8767 -
val_sparse_categorical_crossentropy: 0.8767
```

[ ]: <keras.callbacks.History at 0x7fd0140a6bd0>

[ ]:
```
nn_pred=model.predict([test_roll,test_onehot_data])
```

```
105/105 [==============================] - 1s 6ms/step
```

[ ]:
```
# take argmax in each roll
df = pd.DataFrame(nn_pred)
nn_pred = df.idxmax(axis=1)
```

[ ]:
```
# model reports
generate_model_report(y_test_roll, nn_pred, 'micro')
generate_model_report(y_test_roll, nn_pred, 'macro')
generate_model_report(y_test_roll, nn_pred, 'weighted')
```

```
==================Printing the micro metrics===================
Accuracy =  0.4975
Precision =  0.4975
Recall =  0.4975
F1 Score =  0.4975
=====================================
==================Printing the macro metrics===================
Precision =  0.3303
Recall =  0.3434
F1 Score =  0.3251
=====================================
==================Printing the weighted metrics===================
Precision =  0.4733
Recall =  0.4975
F1 Score =  0.4686
=====================================
```

[ ]:
```
from tensorflow import keras
from kerastuner.tuners import BayesianOptimization, RandomSearch
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Bidirectional,␣
 ↪BatchNormalization
from tensorflow.keras import backend as be
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import keras_tuner
```

```
column_count=len(norm_train_df.columns)
TIME_WINDOW=28
DROPOUT_RATE=0.3
BATCH_SIZE = 32

# function to wrap the model
def build_model(hp):

    be.clear_session()

    float_input = Input(shape=(train_roll.shape[1],train_roll.shape[2] ))
    one_hot_input = Input(shape=(11,) )



    first_lstm = LSTM(units=hp.Int('units',min_value=32,
                                    max_value=64,
                                    step=2))(float_input)
    first_dense = Dense(50)(one_hot_input)
    merge_one = concatenate([first_lstm, first_dense])
    dense_inner = Dense(10)(merge_one)
    dense_output = Dense(3, activation='softmax')(dense_inner)



    model = Model(inputs=[float_input, one_hot_input], outputs=dense_output)
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='Adam',
                  metrics=['sparse_categorical_accuracy'])

    return model

# run random search on the hyperparameters
random_opt_tuner = RandomSearch(
    build_model,
    objective=keras_tuner.Objective('val_sparse_categorical_accuracy','max'),
    # we tried to optimize on f1_macro, but nothing worked, the model didn't␣
 ↪learn with customized f1 function
    max_trials=10,
    executions_per_trial=1,
    project_name='kerastuner_random_poc',
    overwrite=True)

random_opt_tuner.search([train_roll,train_onehot_data], y_roll, epochs=20,␣
 ↪batch_size=BATCH_SIZE,
```

```
                    validation_data = ([test_roll,test_onehot_data],␣
↪y_test_roll),verbose=1)#validation_split=0.2,


random_opt_model_best_model = random_opt_tuner.get_best_models(num_models=1)
model = random_opt_model_best_model[0]
```

```
Trial 10 Complete [00h 03m 01s]
val_sparse_categorical_accuracy: 0.5106255412101746

Best val_sparse_categorical_accuracy So Far: 0.5157138705253601
Total elapsed time: 00h 30m 07s
```

[ ]:
```
from tensorflow.keras.callbacks import EarlyStopping

# fit the untuned model
model_saver = ModelCheckpoint('/content/drive/MyDrive/MADS_23_DL_final_project/
↪data/model_files/LSTM_daily',
                            save_weights_only=True,
                            monitor='val_sparse_categorical_accuracy',
                            mode='max',
                            save_best_only=True)
early_stopping = EarlyStopping(monitor='val_sparse_categorical_accuracy',␣
↪min_delta= 0.001, patience=5, baseline= 0.5)
model.fit([train_roll,train_onehot_data],
          y_roll,
          epochs=50,
          validation_data = ([test_roll, test_onehot_data], y_test_roll),
          batch_size=BATCH_SIZE,
          callbacks=[early_stopping,model_saver])
```

```
Epoch 1/20
527/527 [==============================] - 11s 17ms/step - loss: 0.9230 -
sparse_categorical_accuracy: 0.4904 - val_loss: 0.8682 -
val_sparse_categorical_accuracy: 0.4987
Epoch 2/20
527/527 [==============================] - 8s 15ms/step - loss: 0.9230 -
sparse_categorical_accuracy: 0.4903 - val_loss: 0.8563 -
val_sparse_categorical_accuracy: 0.5037
Epoch 3/20
527/527 [==============================] - 8s 15ms/step - loss: 0.9233 -
sparse_categorical_accuracy: 0.4855 - val_loss: 0.8660 -
val_sparse_categorical_accuracy: 0.4984
Epoch 4/20
527/527 [==============================] - 8s 15ms/step - loss: 0.9224 -
sparse_categorical_accuracy: 0.4923 - val_loss: 0.8648 -
val_sparse_categorical_accuracy: 0.5061
Epoch 5/20
```

```
527/527 [==============================] - 8s 16ms/step - loss: 0.9227 -
sparse_categorical_accuracy: 0.4879 - val_loss: 0.8598 -
val_sparse_categorical_accuracy: 0.5109
Epoch 6/20
527/527 [==============================] - 8s 15ms/step - loss: 0.9226 -
sparse_categorical_accuracy: 0.4899 - val_loss: 0.8689 -
val_sparse_categorical_accuracy: 0.5055
Epoch 7/20
527/527 [==============================] - 8s 16ms/step - loss: 0.9220 -
sparse_categorical_accuracy: 0.4883 - val_loss: 0.8626 -
val_sparse_categorical_accuracy: 0.5073
Epoch 8/20
527/527 [==============================] - 8s 15ms/step - loss: 0.9219 -
sparse_categorical_accuracy: 0.4907 - val_loss: 0.8607 -
val_sparse_categorical_accuracy: 0.4990
Epoch 9/20
527/527 [==============================] - 8s 15ms/step - loss: 0.9219 -
sparse_categorical_accuracy: 0.4895 - val_loss: 0.8613 -
val_sparse_categorical_accuracy: 0.5094
Epoch 10/20
527/527 [==============================] - 8s 15ms/step - loss: 0.9221 -
sparse_categorical_accuracy: 0.4876 - val_loss: 0.8576 -
val_sparse_categorical_accuracy: 0.5052
```

[ ]: <keras.callbacks.History at 0x7fd0191cca10>

[ ]:
```python
# predict with test data and get the argmax
lstm_pred = model.predict([test_roll, test_onehot_data])
df = pd.DataFrame(lstm_pred)
lstm_pred = df.idxmax(axis=1)
```

```
105/105 [==============================] - 1s 5ms/step
```

[ ]:
```python
# model reports
# there's a little bit increase of F1 macro
generate_model_report(y_test_roll, lstm_pred, 'micro')
generate_model_report(y_test_roll, lstm_pred, 'macro')
generate_model_report(y_test_roll, lstm_pred, 'weighted')
```

```
==================Printing the micro metrics===================
Accuracy =  0.5052
Precision =  0.5052
Recall =  0.5052
F1 Score =  0.5052
=====================================
==================Printing the macro metrics===================
Precision =  0.3359
Recall =  0.3501
F1 Score =  0.338
```

```
=====================================
==================Printing the weighted metrics===================
Precision =  0.4814
Recall =  0.5052
F1 Score =  0.4861
=====================================
```

[ ]:

```

```

[ ]:

```
# this is actually the best LSTM we have but didn't get to save the notebook
from tensorflow.keras.models import load_model
model = load_model('/content/drive/MyDrive/MADS_23_DL_final_project/data/
↪model_files/LSTM_daily/LSTM_tw28.h5')
```

[ ]:

```
# predict and get argmax
lstm_pred = model.predict([test_roll, test_onehot_data])
df = pd.DataFrame(lstm_pred)
lstm_pred = df.idxmax(axis=1)
```

```
105/105 [==============================] - 1s 6ms/step
```

[ ]:

```
# model reposts
generate_model_report(y_test_roll, lstm_pred, 'micro')
generate_model_report(y_test_roll, lstm_pred, 'macro')
generate_model_report(y_test_roll, lstm_pred, 'weighted')
```

```
==================Printing the micro metrics===================
Accuracy =  0.5016
Precision =  0.5016
Recall =  0.5016
F1 Score =  0.5016
=====================================
==================Printing the macro metrics===================
Precision =  0.3337
Recall =  0.349
F1 Score =  0.3405
=====================================
==================Printing the weighted metrics===================
Precision =  0.4783
Recall =  0.5016
F1 Score =  0.4887
=====================================
```

[ ]:

```

```

6.1 End of notebook

[ ]:

```

```

[ ]:

```

```