5_model_Testing_F13

November 18, 2022

0.1 5_model_building_tuning_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: We will compare different model outputs .

- 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 the data has.

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:

[ ]:
```
! pip install ta
! pip install seglearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting ta
  Downloading ta-0.10.2.tar.gz (25 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from ta) (1.21.6)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages
(from ta) (1.3.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-
packages (from pandas->ta) (2022.6)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas->ta) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.7.3->pandas->ta) (1.15.0)
Building wheels for collected packages: ta
  Building wheel for ta (setup.py) … done
  Created wheel for ta: filename=ta-0.10.2-py3-none-any.whl size=29104
```

```
sha256=fee003d18efda7b69f39d511464aa9b9d03685c1a10725d8d994e34391f8c0be
  Stored in directory: /root/.cache/pip/wheels/31/31/f1/f2ff471bbc5b84a4b973698c
eecdd453ae043971791adc3431
Successfully built ta
Installing collected packages: ta
Successfully installed ta-0.10.2
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting seglearn
  Downloading seglearn-1.2.5-py3-none-any.whl (11.3 MB)
     |                  | 11.3 MB 3.3 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-
packages (from seglearn) (1.7.3)
Requirement already satisfied: scikit-learn>=0.21.3 in
/usr/local/lib/python3.7/dist-packages (from seglearn) (1.0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from seglearn) (1.21.6)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn>=0.21.3->seglearn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.21.3->seglearn)
(3.1.0)
Installing collected packages: seglearn
Successfully installed seglearn-1.2.5
```

1 Load and transform data

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

```
Mounted at /content/drive
```

[ ]:
```python
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

[ ]:
```

```

2 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')
```

2.1 Code to normalization.

```
[ ]: # from sklearn.preprocessing import MinMaxScaler
     # import warnings
     # from sklearn.exceptions import DataConversionWarning
     # warnings.filterwarnings(action='ignore', category=DataConversionWarning)

     # def minmax_scale(df_x, normalizers=None):


     #     #Try to pass the column values as a parameter from outside
     #     features_to_minmax = ['year','Open','High', 'Low','Close','2h_lag',␣
      ↪'MovingAvg_2hour',
     #         'ExpMovingAvg_2hour', '4h_lag', 'MovingAvg_4hour',␣
      ↪'ExpMovingAvg_4hour',
     #         '6h_lag', 'MovingAvg_6hour', 'ExpMovingAvg_6hour', '8h_lag',
     #         'MovingAvg_8hour', 'ExpMovingAvg_8hour', '10h_lag', 'MovingAvg_10hour',
     #         'ExpMovingAvg_10hour', '12h_lag', 'MovingAvg_12hour',
     #         'ExpMovingAvg_12hour', '14h_lag', 'MovingAvg_14hour',
     #         'ExpMovingAvg_14hour', '16h_lag', 'MovingAvg_16hour',
     #         'ExpMovingAvg_16hour', '18h_lag', 'MovingAvg_18hour',
     #         'ExpMovingAvg_18hour', '20h_lag', 'MovingAvg_20hour',
     #         'ExpMovingAvg_20hour', 'Total_Value',
     #         'Total_Value_market']

     #     if not normalizers:
     #         normalizers = {}

     #     for feat in features_to_minmax:
     #         if feat not in normalizers:
     #             normalizers[feat] = MinMaxScaler()
     #             normalizers[feat].fit(df_x[feat].values.reshape(-1, 1))

     #         df_x[feat] = normalizers[feat].transform(df_x[feat].values.
      ↪reshape(-1, 1))

     #     # series_y=normalizers["pct_change_2hour"].transform(series_y.values.
      ↪reshape(-1, 1))

     #     return df_x , normalizers
```

```
[ ]: # norm_train_df, normalizers = minmax_scale(train_df)
     # norm_test_df, _ = minmax_scale(test_df)
```

```
[ ]: # exporting the dataframe to csv
     # folder_path = '/content/drive/MyDrive/MADS_23_DL_final_project'
```

```
# master_minute_df.to_csv(folder_path + "/data/crypto_data_minute_cleaned_v2.
 ↪csv", index=None)
```

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

3 Calculate percentage change

[ ]:
```
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
```

4 Generate new features

4.0.1 Lag features, moving average, exponential moving average and market cap

[ ]:
```
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
```

[ ]:
```
def create_shift_features(df, col = 'pct_change_1day',lags=10, freq='daily'):

  if freq=='daily':
    mul_fact = 1
    symbol = 'd'
  elif freq=='weekly':
    mul_fact = 7
    symbol = 'W'
  elif freq=='monthly':
    mul_fact = 31
    symbol = 'mon'
  else:
    # setting default to daily
    mul_fact = 1
    symbol = 'd'

  for iterator in range(1,lags+1):

    df['{}_{}_lag'.format(iterator, symbol)] = df[col].
↪shift(periods=iterator*mul_fact)
    # df.loc[:,"Volatility_{}_{}".format(iterator, symbol)] = df[col].
↪rolling(iterator*mul_fact).std().shift(1)


  return df
```

[ ]:
```
#list to collect all relevant lags
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',lags=5, freq='weekly')
    temp_df = create_shift_features(temp_df.copy(),col =␣
↪'pct_change_1day',lags=2, freq='monthly')
```

```
    temp_df = create_shift_features(temp_df.copy(),col =␣
↪'pct_change_1day',lags=4, freq='weekly')
    temp_df = create_shift_features(temp_df.copy(),col =␣
↪'pct_change_1day',lags=12, freq='daily')
    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
```

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

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

[ ]:
```
Index(['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'train_test',
       'Crypto', 'pct_change_1day', '1_mon_lag',
       …
       'momentum_ppo', 'momentum_ppo_signal', 'momentum_ppo_hist',
       'momentum_pvo', 'momentum_pvo_signal', 'momentum_pvo_hist',
       'momentum_kama', 'others_dr', 'others_dlr', 'others_cr'],
      dtype='object', length=113)
```

[ ]:
```
train_df =create_market_volumn_features(train_df.copy())
test_df =create_market_volumn_features(test_df.copy())
```

[ ]:
```
train_df.columns[:50]
```

[ ]:
```
Index(['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'train_test',
       'Crypto', 'pct_change_1day', '1_mon_lag', '2_mon_lag', '1_W_lag',
       '2_W_lag', '3_W_lag', '4_W_lag', '1_d_lag', '2_d_lag', '3_d_lag',
       '4_d_lag', '5_d_lag', '6_d_lag', '7_d_lag', '8_d_lag', '9_d_lag',
       '10_d_lag', '11_d_lag', '12_d_lag', 'volume_adi', 'volume_obv',
       'volume_cmf', 'volume_fi', 'volume_em', 'volume_sma_em', 'volume_vpt',
       'volume_vwap', 'volume_mfi', 'volume_nvi', 'volatility_bbm',
       'volatility_bbh', 'volatility_bbl', 'volatility_bbw', 'volatility_bbp',
       'volatility_bbhi', 'volatility_bbli', 'volatility_kcc',
       'volatility_kch', 'volatility_kcl', 'volatility_kcw', 'volatility_kcp',
       'volatility_kchi'],
      dtype='object')
```

[ ]:
```
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]
#volatility_cols = [col for col in train_df.columns if 'Volatility' in col]
feat_to_keep.extend(lag_cols)
#feat_to_keep.extend(volatility_cols)
```

[ ]:
```
len(feat_to_keep)
```

[ ]: 40

[ ]:
```
train_df = train_df[feat_to_keep]
test_df = test_df[feat_to_keep]
```

[ ]:
```
len(train_df.columns)
```

[ ]: 40

[ ]:
```
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
```

[ ]:
```
train_df = shift_vol(train_df.copy())
test_df = shift_vol(test_df.copy())
```

[ ]:
```

```

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

5.0.1 Encoding of ordinals

[ ]:
```
def encode_cyclicals(df_x):
    '''
    The function converts the date features encoded in the Sine and cosines.
    Input :
    df_x : Input data frame to be processed
    Output :
    df_x : processed dataframe.
    '''
     #"month","day","hour", "minute", "dayofweek"



     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
```

[ ]:
```
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
```

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

5.1 One hot coding the coins

By this process we are tagging which record belongs which coin

[ ]:
```
# 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)
    df = pd.concat([df, y_dummies], axis=1)
    df.drop(['Crypto'], axis=1, inplace=True)
    # creating a additional column if the model is used for new coin.
    df['other_crypto'] =0
    return df
```

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

[ ]:
```
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
```

5.1.1 Defining the Target Variable.

We want to follow the classification approach and hence based on the "pct_change_2hour" we are creating 3 classes one class '0' when the returns are negative and '1' When the retruns are postive.

Finally, 2 when the returns beat the market value.

[ ]:
```
def create_target(df):
  market_RoR = 26.89
  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)
  return df
```

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

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

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

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

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

[ ]:
```
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
```

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

[ ]:
```
(17125, 59)
```

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

[ ]:
```
(3621, 59)
```

[ ]:
```
target = train_df['Target']
test_target = test_df['Target']
```

[ ]:
```

```

6 Drop 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)
```

[ ]:
```
#plt.figure(figsize=(30,30))

#corr = train_df.drop(['Open','High', 'Low','Close','Crypto_ADA',
#        'Crypto_BTC', 'Crypto_ETC', 'Crypto_ETH', 'Crypto_LINK', 'Crypto_LTC',
#        'Crypto_TRX', 'Crypto_XLM', 'Crypto_XMR', 'Crypto_XRP',␣
 ↪'other_crypto'],axis=1).corr()

#sns.heatmap(corr,vmin=0, vmax=1, annot=True, cmap="YlGnBu")
```

[ ]:
```
# dropping the listof the columns
# drop_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', '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_mon_lag', '2_mon_lag',
       '1_W_lag', '2_W_lag', '3_W_lag', '4_W_lag', '1_d_lag', '2_d_lag',
       '3_d_lag', '4_d_lag', '5_d_lag', '6_d_lag', '7_d_lag', '8_d_lag',
       '9_d_lag', '10_d_lag', '11_d_lag', '12_d_lag', 'year', 'month_sin',
       'month_cos', 'day_sin', 'day_cos', 'dayofweek_sin', 'dayofweek_cos',
       'hour_sin', 'hour_cos', 'Crypto_ADA', 'Crypto_BTC', 'Crypto_ETC',
       'Crypto_ETH', 'Crypto_LINK', 'Crypto_LTC', 'Crypto_TRX', 'Crypto_XLM',
       'Crypto_XMR', 'Crypto_XRP', 'other_crypto'],
      dtype='object')
```

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

[ ]:
```
(17125, 54)
```

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

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

```
[ ]: # metrics for the classification
     # from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
      ↪f1_score
     # def generate_model_report(y_actual, y_predicted):
     #     print("Accuracy = " , accuracy_score(y_actual, y_predicted))
     #     print("Precision = " ,precision_score(y_actual, y_predicted))
     #     print("Recall = " ,recall_score(y_actual, y_predicted))
     #     print("F1 Score = " ,f1_score(y_actual, y_predicted))
```

```
[ ]:
```

7 Evaluation function

```
[ ]: import sklearn
     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(sklearn.metrics.f1_score(y_actual, y_predicted,␣
      ↪average=metric_type),4))
         print("=====================================")
         return round(sklearn.metrics.f1_score(y_actual, y_predicted,␣
      ↪average=metric_type),4)
```

```
[ ]: !pip install tensorflow-addons
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.18.0-cp37-cp37m-manylinux_2_17_x86_64.manylinu
x2014_x86_64.whl (1.1 MB)
     |                 | 1.1 MB 5.3 MB/s
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)
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.18.0
```

[ ]:
```
master_df = pd.DataFrame(columns=[daily['Crypto'].unique()])



master_df.loc['Dummy'] = 1
master_df.loc['XGboost'] = 1
master_df.loc['LGBM'] = 1
master_df.loc['LSTM'] = 1
```

[ ]:

[ ]:

[ ]:
```
master_df.loc['LSTM', 'BTC'] = 25
```

[ ]:
```
master_df
```

[ ]:

	BTC	ETH	XRP	ADA	LTC	LINK	XLM	TRX	XMR	ETC
Dummy	1	1	1	1	1	1	1	1	1	1
XGboost	1	1	1	1	1	1	1	1	1	1
LGBM	1	1	1	1	1	1	1	1	1	1
LSTM	25	1	1	1	1	1	1	1	1	1

[ ]:
```
def model_predic(algo , norm_test_df):


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


  for coin in crypto_coins:


    # print(coin)
    coin_index = norm_test_df[norm_test_df['Crypto_'+coin]==1]
    master_df.loc[algo, coin] = sklearn.metrics.f1_score(coin_index['pred'],
 ↪coin_index['actual'], average='macro')

  # return master_df_x, master_df_y
```

7.1 Dummy Predictor

[ ]:
```
# Reading the file
import pickle
temp_file_path = '/content/drive/MyDrive/MADS_23_DL_final_project/data/
 ↪model_files/xgboost_hourly'
# loading the pickled model
with open(temp_file_path + '/dummy_f13_final.pkl', 'rb') as f:  # Python 3:␣
 ↪open(..., 'wb')
    dummy_model = pickle.load(f)
```

[ ]:
```
norm_test_df['pred']=dummy_model.predict(norm_test_df.fillna(0))

norm_test_df['actual'] = test_target
model_predic('Dummy',norm_test_df)
```

[ ]:
```
master_df
```

[ ]:
```
              BTC       ETH       XRP       ADA       LTC      LINK       XLM  \
Dummy    0.227437  0.225455  0.218905  0.227465  0.221814  0.215174  0.218477
XGboost         1         1         1         1         1         1         1
LGBM            1         1         1         1         1         1         1
LSTM            1         1         1         1         1         1         1

              TRX       XMR       ETC
Dummy    0.188579  0.219331  0.229844
XGboost         1         1         1
LGBM            1         1         1
LSTM            1         1         1
```

7.2 Xgboost

[ ]:
```
# Reading the file
import pickle
temp_file_path = '/content/drive/MyDrive/MADS_23_DL_final_project/data/
 ↪model_files/xgboost_hourly'
# loading the pickled model
with open(temp_file_path + '/XGBoost_f13_final.pkl', 'rb') as f:  # Python 3:␣
 ↪open(..., 'wb')
    XGboost_model = pickle.load(f)
```

[ ]:
```
norm_test_df['pred']=XGboost_model.predict(norm_test_df.fillna(0))

norm_test_df['actual'] = test_target
model_predic('XGboost',norm_test_df)
```

[ ]:
```
# Reading the file
import pickle
temp_file_path = '/content/drive/MyDrive/MADS_23_DL_final_project/data/
↪model_files/xgboost_hourly'
# loading the pickled model
df2 = pd.read_csv(temp_file_path + "/dummy_cs.csv")
```

[ ]:
```
df2.head()
```

[ ]:
```
        BTC       ETH       XRP       ADA       LTC     LINK       XLM  \
0  1.000000  1.000000  1.000000  1.000000  1.000000  1.00000  1.000000
1  0.335962  0.368582  0.375731  0.390007  0.401261  0.38179  0.419516
2  1.000000  1.000000  1.000000  1.000000  1.000000  1.00000  1.000000
3  1.000000  1.000000  1.000000  1.000000  1.000000  1.00000  1.000000

        TRX      XMR       ETC
0  1.000000  1.00000  1.000000
1  0.446049  0.34036  0.387906
2  1.000000  1.00000  1.000000
3  1.000000  1.00000  1.000000
```

[ ]:
```
master_df.loc['XGboost'] = df2.loc[1].values
```

[ ]:
```

```

[ ]:
```
array([0.33596214, 0.36858155, 0.37573145, 0.39000684, 0.40126067,
       0.38178963, 0.41951601, 0.44604911, 0.34035966, 0.38790629])
```

[ ]:
```
master_df
```

[ ]:
```
              BTC       ETH       XRP       ADA       LTC     LINK       XLM  \
Dummy    0.227437  0.225455  0.218905  0.227465  0.221814  0.215174  0.218477
XGboost  0.335962  0.368582  0.375731  0.390007  0.401261   0.38179  0.419516
LGBM            1         1         1         1         1         1         1
LSTM            1         1         1         1         1         1         1

              TRX       XMR       ETC
Dummy    0.188579  0.219331  0.229844
XGboost  0.446049   0.34036  0.387906
LGBM            1         1         1
LSTM            1         1         1
```

7.3 LGBM

[ ]:
```
# Reading the file
import pickle
temp_file_path = '/content/drive/MyDrive/MADS_23_DL_final_project/data/
↪model_files/LGBM_hourly'
```

```
# loading the pickled model
with open(temp_file_path + '/Tuned_lgbm_v13_final2.pkl', 'rb') as f:  # Python␣
↪3: open(..., 'wb')
    untuned_lgbm = pickle.load(f)
```

[ ]:
```
norm_test_df.columns
```

[ ]:
```
Index(['High', 'Low', 'Volume', '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_mon_lag', '2_mon_lag',
       '1_W_lag', '2_W_lag', '3_W_lag', '4_W_lag', '1_d_lag', '2_d_lag',
       '3_d_lag', '4_d_lag', '5_d_lag', '6_d_lag', '7_d_lag', '8_d_lag',
       '9_d_lag', '10_d_lag', '11_d_lag', '12_d_lag', 'year', 'month_sin',
       'month_cos', 'day_sin', 'day_cos', 'dayofweek_sin', 'dayofweek_cos',
       'hour_sin', 'hour_cos', 'Crypto_ADA', 'Crypto_BTC', 'Crypto_ETC',
       'Crypto_ETH', 'Crypto_LINK', 'Crypto_LTC', 'Crypto_TRX', 'Crypto_XLM',
       'Crypto_XMR', 'Crypto_XRP', 'other_crypto', 'pred', 'actual'],
      dtype='object')
```

[ ]:
```
norm_test_df['pred']=untuned_lgbm.predict(norm_test_df[['High', 'Low',␣
↪'Volume', '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_mon_lag', '2_mon_lag',
       '1_W_lag', '2_W_lag', '3_W_lag', '4_W_lag', '1_d_lag', '2_d_lag',
       '3_d_lag', '4_d_lag', '5_d_lag', '6_d_lag', '7_d_lag', '8_d_lag',
       'year', 'month_sin', 'month_cos', 'day_sin', 'day_cos', 'dayofweek_sin',
       'dayofweek_cos', 'hour_sin', 'hour_cos', 'Crypto_ADA', 'Crypto_BTC',
       'Crypto_ETC', 'Crypto_ETH', 'Crypto_LINK', 'Crypto_LTC', 'Crypto_TRX',
       'Crypto_XLM', 'Crypto_XMR', 'Crypto_XRP', 'other_crypto']].fillna(0))

norm_test_df['actual'] = test_target
model_predic('LGBM',norm_test_df)
```

[ ]:
```
temp_file_path = "/content/drive/MyDrive/MADS_23_DL_final_project/data/
↪model_files/"
df2 = pd.read_csv(temp_file_path + "/dummy_.csv")
```

[ ]:
```
df2
```

[ ]:
```
        BTC       ETH       XRP       ADA       LTC      LINK       XLM  \
0  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
```

```
1  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
2  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
3  0.289043  0.202899  0.230112  0.322767  0.322533  0.224422  0.247148

        TRX       XMR       ETC
0  1.000000  1.000000  1.000000
1  1.000000  1.000000  1.000000
2  1.000000  1.000000  1.000000
3  0.277402  0.211749  0.331912
```

[ ]:
```
master_df.loc['LSTM'] = df2.loc[3].values
```

[ ]:
```
master_df
```

[ ]:
```
              BTC       ETH       XRP       ADA       LTC      LINK       XLM  \
Dummy    0.227437  0.225455  0.218905  0.227465  0.221814  0.215174  0.218477
XGboost  0.335962  0.368582  0.375731  0.390007  0.401261   0.38179  0.419516
LGBM     0.325749  0.343658  0.371659  0.372179  0.396585  0.415688  0.389261
LSTM     0.289043  0.202899  0.230112  0.322767  0.322533  0.224422  0.247148

              TRX       XMR       ETC
Dummy    0.188579  0.219331  0.229844
XGboost  0.446049   0.34036  0.387906
LGBM     0.429581  0.346284  0.367134
LSTM     0.277402  0.211749  0.331912
```

[ ]:
```
master_df = master_df.T
```

[ ]:
```
import plotly.express as px
fig = px.line(master_df, y=master_df.columns, x = master_df.index).
 ↪update_layout(
    xaxis_title="Coins", yaxis_title="F1 Score(macro)",  width=600, height=400
)

# Show plot

fig.show()
```

Observation: Overall the XGBoost and LGBM are doing better than rest of the algoritms. Since it is a classification problem the conventonal model are doing better than the Neural network Models.

Amoung the coins the model is able to best perform for the TRX coin.

7.4 End of the Notebook

[ ]:
```
```