NN Forecast

Imports

Importig Two helper functions from `./helper_functions`:

- `stock_list`: This functions gets an index name (e.g. ‘Dow Jones’, ‘CAC 40’, ‘DAX’, ‘Teh50’) and returns the list of stocks in that index.
- `stock_prices`: This functions recieves a list of tickers and returns a pandas dataframe containing prices of the corresponding tickers.

[ ]:

```python
from helper_functions import stock_prices, stock_list
```

[ ]:

```python
import numpy as np
import pandas as pd

from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.linear_model import Ridge
from sklearn.neural_network import MLPRegressor
import matplotlib.pyplot as plt
```

Constants and Fucntions

We will work with the 521 trailing days of prices series. 126 last days will be the test period.

[ ]:

```python
interval = 521
testsmpl=126
```

Helper Functions

This function recives a dataframe and returns the lag matrix of its first column.

[ ]:

```python
def lagmat(df, T=20) -> (np.array, np.array):
    X = []
    Y = []
    tag = df.columns[0]
    series = df[tag].to_numpy()[:]
    for t in range(len(series) - T):
        x = series[t:t+T]
        X.append(x)
        y = series[t+T]
        Y.append(y)

    X = np.array(X).reshape(-1 ,T)
    Y = np.array(Y)

    return X, Y
```

This helper function Will convert the arabic glyphs to standard farsi glyphs. This will be helpful while looking Tehran50 tickers up:

[ ]:

```
def groom(s):
    s = s.replace(' ', ' '(
    s = s.replace(' ', ' '(
    return s
```

This functino calculates the MAPE of two seres. It receives a dataframe that has two columns: actual and forecasted values.

[ ]:

```
from sklearn.metrics import mean_absolute_percentage_error
def get_mape(df, ticker, pred_tag, test_count=126):
    df = df.dropna(how='any')
    test_true = df.iloc[-test_count:][ticker]
    test_pred = df.iloc[-test_count:][f'{ticker}_{pred_tag}']
    mape = mean_absolute_percentage_error(
        test_true, test_pred
    )
    return mape
```

Because running this code may take a long time, we save the result of each model when it's done. This functino opens and excel file and append the last calculated mape values:

[ ]:

```
import os
def write_to_excel(errors):
    filename = rf'./nn_mape.xlsx'
    if not os.path.exists(filename):
        writer = pd.ExcelWriter(filename, engine='openpyxl', mode='w')
    else:
        writer = pd.ExcelWriter(filename, engine='openpyxl', mode='a',␣
 ↪if_sheet_exists='overlay')
    df_errors = pd.DataFrame(errors)
    for index, group_df in df_errors.groupby("indice"):
        group_df.to_excel(writer, sheet_name=str(index),index=False)
    writer.save()
```

We read the list of cointegrated tickers from the output of the previous project (i.e. 01_pair_trading).

[ ]:

```
df_indices = pd.DataFrame()
file = pd.ExcelFile('../01_pair_trading/pairs_2023-01-15.xlsx')
sheet_names = ['Dow Jones', 'CAC 40', 'Dax', 'Teh50']
for sheet in sheet_names:
    df_tmp = pd.read_excel(file, sheet_name=sheet)
    df_indices = df_indices.append(df_tmp)
file.close()
```

Forecast

We explain the code in 7 steps. The starting point of each step is commented in the code by the corresponding number:

1. We Will save all the plots in the `./preds_nn/{index_name}/` directory. We will remove and recreate the directory each time we run the code.
2. We write a for loop that itterates over four indices: 'Dow Jones', 'CAC 40', 'Dax', 'Teh50'
3. We get the list of tickers that are cointegrated from the previous project.

4. We get the price of all the cointegrated tickers in each index. We split the data into train and test.
5. **Dynamic Multi-Step ahead forecast**: We build a MLPRegressor model based on the first 521 days and forecast the 126 days ahead. MLPRegressor class uses Dynamic forecasting by default.
6. **One-Step ahead forecast**: For each day in the test period, we consider all leading days as the training set and build a MLPRegressor model based on the training data and forecast one step ahead.
7. We plot the the actual values in conjunction with the forecasted values in one graph and save them in `./preds_vecm/{index_name}/` directory.
8. We save the mape value of our forecast in a dictinoary. The will later be used to comapre with VECM's forecasts.

[ ]:

```
import itertools
import os
# 1
PATH = r'./preds_nn'
if not os.path.exists(PATH):
    os.makedirs(PATH)

errors = []

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

# 2
for indice in ['Dow Jones', 'CAC 40', 'Dax', 'Teh50']:
    # Because the
    try:
        df_dones = pd.read_excel('nn_mape2.xlsx', sheet_name=indice)
    except:
        df_dones = pd.DataFrame([], columns=['tag', 'ticker', 'pair', 'mape',␣
 ↪'indice'])

    print(indice, '>>', flush=True)
    # Creating the required directories to save the plots
    PATH = rf'./preds_nn/{indice}/'
    if not os.path.exists(PATH):
        os.makedirs(PATH)

    # 3
    df1 = df_indices[df_indices['indice']==indice]
    tickers = stock_list.get_stock_list(index=indice)
    isTSE = (indice == 'Teh50')
    if isTSE:
        tickers = [groom(x) for x in tickers]
    data_historical = stock_prices.get_prices(tickers, isTSE)

    for i in range(df1.shape[0]):
        print(f'{i}', end=',', flush=True)
        ticker1, ticker2, indice = df1.iloc[i]

        if df_dones[(
            (df_dones['ticker']==ticker1)
            & (df_dones['pair']==ticker2)
        )].shape[0] > 0:
            continue

        # 4
        data_historical1 = data_historical[[ticker1, ticker2]]
        data_historical1 = data_historical1.dropna(how='all')
        data = data_historical1[-interval:]
        limitPer = len(data) * .85
        data = data.dropna(thresh=limitPer, axis=1)
        data = np.log(data)
        data = data.dropna(how='any')
        X1, Y1 = lagmat(data[[ticker1]])
        X2, Y2 = lagmat(data[[ticker2]])
        Y1 = np.expand_dims(Y1, axis=1)
        Y2 = np.expand_dims(Y2, axis=1)
        X = np.hstack((X1, X2))
        Y = np.hstack((Y1, Y2))
        X_train, X_test = X[:-testsmpl], X[-testsmpl:]
        Y_train, Y_test = Y[:-testsmpl], Y[-testsmpl:]
        data_train = data[:-testsmpl]
        data_test = data[-testsmpl:]
        df_train = data_train.copy()
        df_data = pd.DataFrame([], columns=[ticker1, ticker2])
        df_preds = pd.DataFrame([], columns=[f'{ticker1}_multi', f'{ticker2}_multi',
        f'{ticker1}_1step', f'{ticker2}_1step'])
        data[df_preds.columns] = np.nan

        # 5
        preds_regr = MultiOutputRegressor(MLPRegressor()).fit(X_train, Y_train)
        forecast_multistep = preds_regr.predict(X_test)
        df_preds[[f'{ticker1}_multi', f'{ticker2}_multi']] = forecast_multistep

        # 6
        preds_1step = np.ndarray(shape=(0,2))
        X_train_1step = X_train.copy()
        Y_train_1step = Y_train.copy()
        preds_regr = MultiOutputRegressor(MLPRegressor()).fit(X_train, Y_train)
        for i in range(X_test.shape[0]):
            forecast_1step = preds_regr.predict([X_test[i]])
            preds_1step = np.vstack([preds_1step, forecast_1step])

            X_train_1step = np.vstack((X_train_1step, X_test[i]))
            Y_train_1step = np.vstack([Y_train_1step, Y_test[i]])
        df_preds[[f'{ticker1}_1step', f'{ticker2}_1step']] = preds_1step
        data.iloc[-testsmpl:, 2:] = df_preds

        # 7
        # Plotting
        ax = data.plot(figsize=(15, 8));
        ax.figure.savefig(rf'./preds_nn/{indice}/{ticker1}_{ticker2}.png');
        plt.close()



        # 8
        # mape
        errors = []
        for ticker, tag in list(itertools.product([ticker1, ticker2], ['1step',␣
↪'multi'])):
            mape=get_mape(data, ticker=ticker, pred_tag=tag)
            errors.append({
                'tag': f'nn_{tag}',
                'ticker': ticker,
                'pair': ticker2 if ticker==ticker1 else ticker1,
                'mape': mape*100,
                'indice': indice
            })
        df_dones = df_dones.append(errors, ignore_index=True)
        write_to_excel(df_dones)
```