0.2.PredictUsingModel

December 20, 2023

This notebook demonstrates prediction using pretrained model

0.0.1 Importing modules

[1]:
```
import pickle
import numpy as np
from src.loader import ModelLoader
from src.postprocessor import invert_scale_N_feature, calculate_rmse
import matplotlib.pyplot as plt
```

0.0.2 Set up

Let's specify *ticker* and *version* assuming that the data version and model version are the same. We can always get a fresh set of data and train model using that as shown in $0.1.ModelTrainingWalk-through.ipynb$

[2]:
```
ticker = 'QQEW'
version = 0
rolling_window = 20
forecast_horizon = 20
```

0.0.3 Contruct model name

Without explicitly specifying model name, we can *construct model name* using syntax. > Syntax: TICKER_MODELFAMILY_RollingWindow_ForecastHorizon > Example: QQEW_LSTM_RW20_FH20_V2

[3]:
```
basename = '{}_LSTM_RW{}_FH{}_V{}'.format(ticker, rolling_window,␣
 ↪forecast_horizon, version)
print('basename =>', basename)
```

```
basename => QQEW_LSTM_RW20_FH20_V0
```

The same basename will be used to read model training history

0.0.4 Get the dataset

We have saved all the data in pickle file before training model. The filename schema remains similar. Let's read that.

[4]:
```
filename = 'data/{}_PREPROCESSED_V{}.pickle'.format(ticker, version)
print('loading dataset from =>', filename)

with open(filename, 'rb') as f:
    dataset = pickle.load(f)
    print('dataset loaded')
```

```
loading dataset from => data/QQEW_PREPROCESSED_V0.pickle
dataset loaded
```

The dataset object is a python dict. Let's see the keys.

[5]:
```
dataset.keys()
```

[5]:
```
dict_keys(['splits', 'normalised_train', 'normalised_test', 'training_set',
'test_set'])
```

From dataset, we need the followings: > splits: > > we stored df_predict here

normalized_test: > we stored scalers of test set here

test_set: > we stored features and labels for testing here

Dataset is a nested dict. Let's access inside the key.

[6]:
```
for key in dataset.keys():
    print(key, '--' ,dataset[key].keys())
```

```
splits -- dict_keys(['df_train', 'df_test', 'df_predict'])
normalised_train -- dict_keys(['normalised_data_py_list', 'scalers'])
normalised_test -- dict_keys(['normalised_data_py_list', 'scalers'])
training_set -- dict_keys(['normalised_data_np_array', 'features', 'labels'])
test_set -- dict_keys(['normalised_data_np_array', 'features', 'labels'])
```

0.0.5 Load Model

Now, we can instantiate *ModelLoader* class and load the model using *read_model_local* method.

[7]:
```
model = ModelLoader()
```

We have tiny docstring for help text. > Docstrings are not complete at this moment.

[8]:
```
help(model.read_model_local)
```

```
Help on method read_model_local in module src.loader:

read_model_local(basename=None) method of src.loader.ModelLoader instance
    reads model specified by basename from /model directory of project root
```

Read the model!

[9]:
```
model.read_model_local(basename=basename)
```

```
Loading model from model/QQEW_LSTM_RW20_FH20_V0.h5
```

Model can be accessed in ModelLoader attribute as *model.model*

Let's read model training history as dict object. It can be accessed as *training_history* attribute of *ModelLoader* class.

[10]:
```
model.read_training_history(basename=basename)
model.training_history.keys()
```

[10]:
```
dict_keys(['loss', 'val_loss', 'lr'])
```

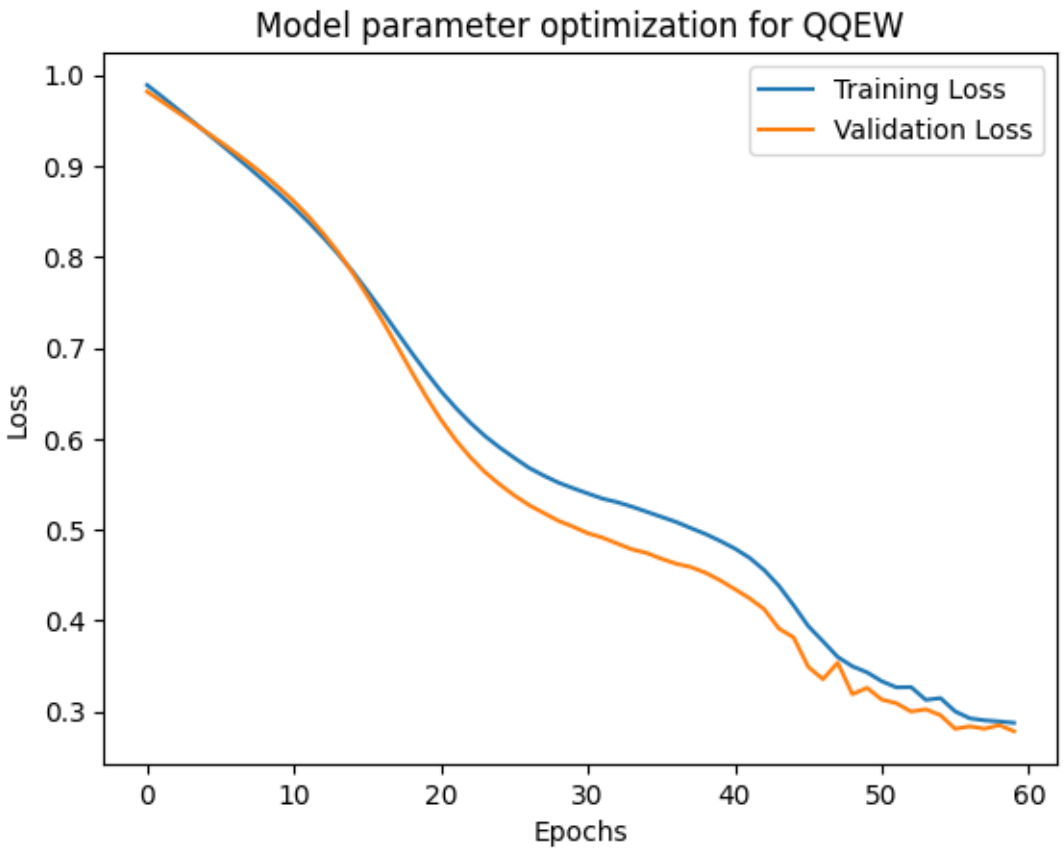
In *training_history*, > *loss* stands for loss in training set > *val_loss* stands for loss in validation set

We have used test_set as validation set during model training. We will use prediction set to test the model's final accuracy. » We are having notation assignment problem here.

0.0.6 Plot

Let's plot loss in training set and loss in validation set.

[11]:
```
plt.plot(model.training_history['loss'], label='Training Loss')
plt.plot(model.training_history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
leg = plt.legend(loc='upper right')
plt.title('Model parameter optimization for QQEW')
plt.show()
```



[ ]:

[ ]:

0.0.7 Calculated Error (RMSE) in test set

[12]:
```
# # This cell is to figure out keys
# for key in dataset.keys():
#     print(key, '--' ,dataset[key].keys())
```

[13]:
```
# This cell is to check test set shape
print('test_set => normalised_data_np_array => shape => ',␣
 ↪dataset['test_set']['normalised_data_np_array'].shape)
print('test_set => features => shape => ', dataset['test_set']['features'].
 ↪shape)
```

```
test_set => normalised_data_np_array => shape =>  (1269, 40, 19)
test_set => features => shape =>  (1269, 20, 18)
```

Run predictions on test set

[14]:
```
predictions = model.model.predict(dataset['test_set']['features'])
```

```
40/40 [==============================] - 5s 114ms/step
```

[15]:
```
# # check more shapes
# print(len(dataset['normalised_test']['scalers']))
# print(dataset['test_set']['normalized_data_np_array'].shape)
# print(dataset['test_set']['labels'].squeeze().shape)
# print(predictions.squeeze().shape)
```

RMSE in USD

[16]:
```
rmse_in_usd = calculate_rmse(
    scalers=dataset['normalised_test']['scalers'],
    data=dataset['test_set']['normalised_data_np_array'],
    labels=dataset['test_set']['labels'].squeeze(),
    predicts=predictions.squeeze(), splitname='test')
```

```
The avearge root mse on test data is: 1.29
```

[17]:
```
# rmse_in_usd
```

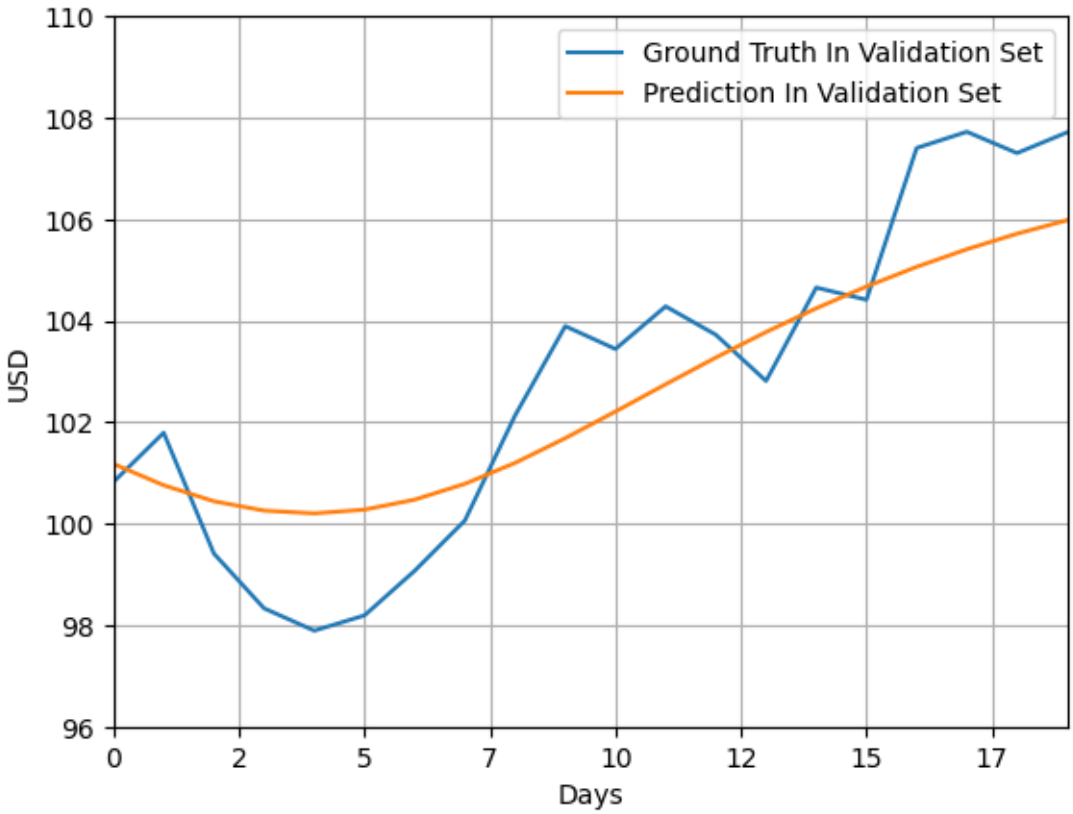
0.0.8 Visualize last sample from test set

[18]:
```
idx = len(dataset['normalised_test']['scalers']) - 1
# what the date for last sample?
# TODO: needs to implement a function to find date using prediction set

[inv_true, inv_pred] = invert_scale_N_feature(
    scaler=dataset['normalised_test']['scalers'][idx],
    data=dataset['test_set']['normalised_data_np_array'][idx],
    prediction=predictions.squeeze()[idx]
)
```

Plot prediction results in test set

[19]:
```
plt.plot(inv_true[-rolling_window:], label='Ground Truth In Validation Set')
plt.plot(inv_pred, label='Prediction In Validation Set')
plt.xlabel('Days')
plt.ylabel('USD')
leg = plt.legend(loc='upper right')
plt.yticks(ticks=plt.yticks()[0], labels=plt.yticks()[0].astype(int))
plt.xticks(ticks=plt.xticks()[0], labels=plt.xticks()[0].astype(int))
plt.xlim(xmin=0, xmax=19)  # this line
plt.grid()
plt.show()
```



0.0.9 The following needs to be wrapped within a class for repeated use

0.0.10 Predict using prediction dataset

[20]:
```
# prediction dataset
dataset['splits']['df_predict']
```

[20]:

Date	Adj Close	HL_PCT	PCT_change	Volume	MA_5	MA_20
2023-11-20	109.029999	1.289424	1.075365	179600	107.832001	103.161501
2023-11-21	108.360001	0.546242	-0.220993	50500	108.024001	103.490001
2023-11-22	108.699997	0.782904	-0.027597	68800	108.220000	103.954500
2023-11-24	109.000000	0.331126	0.183821	34700	108.559999	104.488000
2023-11-27	108.720001	0.617510	-0.137780	53700	108.762000	105.029500
2023-11-28	108.889999	0.626902	0.193227	79700	108.734000	105.564500
2023-11-29	109.529999	0.923051	-0.036507	184900	108.967999	106.087500
2023-11-30	109.610001	0.973994	-0.245720	167200	109.150000	106.565000
2023-12-01	110.779999	1.530564	1.261423	105100	109.506000	106.997500
2023-12-04	110.110001	1.016391	0.081807	126200	109.784000	107.308500
2023-12-05	109.400002	0.762309	0.073181	213700	109.886000	107.606500
2023-12-06	109.239998	1.044817	-0.736034	161400	109.828000	107.854500

```
2023-12-07  110.330002  1.005851    0.363869   73100  109.972000  108.185000
2023-12-08  110.559998  1.085168    0.545652   78400  109.928000  108.572500
2023-12-11  112.639999  1.612178    1.450059  131100  110.434000  108.972000
2023-12-12  113.220001  0.972003    0.461406  137000  111.198000  109.412500
2023-12-13  115.430000  1.942602    1.682526  360900  112.436000  109.814000
2023-12-14  116.300003  1.263303   -0.025788  173000  113.630000  110.243000
2023-12-15  116.150002  0.777002   -0.428633  101200  114.748001  110.685500
2023-12-18  116.410004  0.525947   -0.034344  108500  115.502002  111.120500

                MA_60       EMA_5     up_band    mid_band    low_band  \
Date
2023-11-20  104.579355  104.508000  109.075310  107.832001  106.588691
2023-11-21  104.620254  105.127334  109.237298  108.024001  106.810703
2023-11-22  104.633538  105.703334  109.488886  108.220000  106.951113
2023-11-24  104.640839  106.161334  109.538406  108.559999  107.581592
2023-11-27  104.644638  106.483334  109.248357  108.762000  108.275642
2023-11-28  104.645114  106.846667  109.168805  108.734000  108.299194
2023-11-29  104.664077  107.196667  109.572171  108.967999  108.363827
2023-11-30  104.687037  107.589334  109.860492  109.150000  108.439508
2023-12-01  104.741810  108.120667  110.956990  109.506000  108.055010
2023-12-04  104.786415  108.484667  111.046480  109.784000  108.521520
2023-12-05  104.805043  108.817333  110.901290  109.886000  108.870710
2023-12-06  104.834815  108.940000  110.945955  109.828000  108.710045
2023-12-07  104.884585  109.114000  111.125458  109.972000  108.818542
2023-12-08  104.921214  109.331333  110.965798  109.928000  108.890202
2023-12-11  105.022131  109.660000  112.865118  110.434000  108.002882
2023-12-12  105.133879  109.939333  114.186255  111.198000  108.209744
2023-12-13  105.288452  110.410666  116.185659  112.436000  108.686341
2023-12-14  105.473000  110.917333  117.723120  113.630000  109.536881
2023-12-15  105.686333  111.394000  117.796659  114.748001  111.699343
2023-12-18  105.903333  111.906667  117.884251  115.502002  113.119753

                 ADX      MACD        RSI       ATR       MOM      WILLR  \
Date
2023-11-20  16.257619  0.808405  67.801064  1.479729  5.589996  -2.473008
2023-11-21  16.825209  0.733538  64.454540  1.446891  4.080002 -12.828465
2023-11-22  17.552810  0.663583  65.388154  1.419255  4.979996 -10.746292
2023-11-24  18.244031  0.596310  66.230980  1.345023  6.190002  -6.491470
2023-11-27  18.800600  0.495871  64.648683  1.296807  4.070000 -12.658175
2023-11-28  19.315225  0.407496  65.192401  1.252749  4.479996 -16.562478
2023-11-29  20.120595  0.360382  67.235510  1.273267  2.129997 -25.210129
2023-11-30  20.604605  0.305040  67.492376  1.258034  1.889999 -22.969181
2023-12-01  21.281476  0.316307  71.065076  1.287460  3.479996  -0.000000
2023-12-04  21.924503  0.250588  66.554210  1.307641  2.400002 -22.483192
2023-12-05  22.369192  0.136937  62.058736  1.302096  0.370003 -49.819456
2023-12-06  22.933234  0.033988  61.057819  1.290517  0.879997 -66.666777
2023-12-07  23.523746  0.022838  65.178130  1.285481  1.630005 -19.480408
```

```
2023-12-08  24.186567  0.014876  65.995691  1.278660  1.559998 -12.184926
2023-12-11  25.266153  0.127358  72.324147  1.348756  3.919998  -4.137940
2023-12-12  26.373740  0.213418  73.789011  1.330274  4.330002  -0.227321
2023-12-13  27.812452  0.381943  78.466020  1.394540  5.900002  -0.302064
2023-12-14  29.401716  0.506935  79.980460  1.409215  6.690002  -8.957001
2023-12-15  30.911516  0.531839  78.949512  1.372843  5.370003 -10.797510
2023-12-18  32.345826  0.517658  79.444113  1.318354  6.300003  -7.607301
```

Date	CCI	OBV
2023-11-20	123.175739	10456600.0
2023-11-21	93.614549	10406100.0
2023-11-22	96.026786	10474900.0
2023-11-24	84.952810	10509600.0
2023-11-27	72.870241	10455900.0
2023-11-28	68.749304	10535600.0
2023-11-29	99.768273	10720500.0
2023-11-30	82.629837	10887700.0
2023-12-01	124.293630	10992800.0
2023-12-04	105.656377	10866600.0
2023-12-05	48.964872	10652900.0
2023-12-06	59.986226	10491500.0
2023-12-07	100.926031	10564600.0
2023-12-08	123.928226	10643000.0
2023-12-11	248.200343	10774100.0
2023-12-12	228.538124	10911100.0
2023-12-13	234.100500	11272000.0
2023-12-14	201.999329	11445000.0
2023-12-15	147.934430	11343800.0
2023-12-18	120.756616	11452300.0

[21]:
```
first_date = dataset['splits']['df_predict'].index[0].date().
 ↪strftime('%b-%d-%Y')
last_date = dataset['splits']['df_predict'].index[-1].date().
 ↪strftime('%b-%d-%Y')
print('from {} to {}'.format(first_date, last_date))
```

```
from Nov-20-2023 to Dec-18-2023
```

[22]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

Scaling data for prediction

[23]:
```
scaler.fit(dataset['splits']['df_predict'])

# scaling data
ndata = scaler.transform(dataset['splits']['df_predict'])
```

```
ndata_arr = np.array([ndata])
ndata_arr.shape
```

[23]:
```
(1, 20, 19)
```

Isolating features and label from scaled data

[24]:
```
ndata_arr_feature = ndata_arr[:, :, 1:]
ndata_arr_label = ndata_arr[:,:,:0] # This is not required since the future is␣
 ↪unknown
```

[25]:
```
# ndata_arr_feature.shape
```

Predicting future price

[26]:
```
forecast = model.model.predict(ndata_arr_feature)
```

```
1/1 [==============================] - 0s 22ms/step
```

[27]:
```
forecast.shape
```

[27]:
```
(1, 20, 1)
```

Inversing scale: normalised data to USD

[28]:
```
[_, inv_pred_forecast] = invert_scale_N_feature(
    scaler=scaler,
    data=ndata_arr[0],
    prediction=forecast.squeeze()
) # we do not have any true inverse here. TODO: Implement new version
```

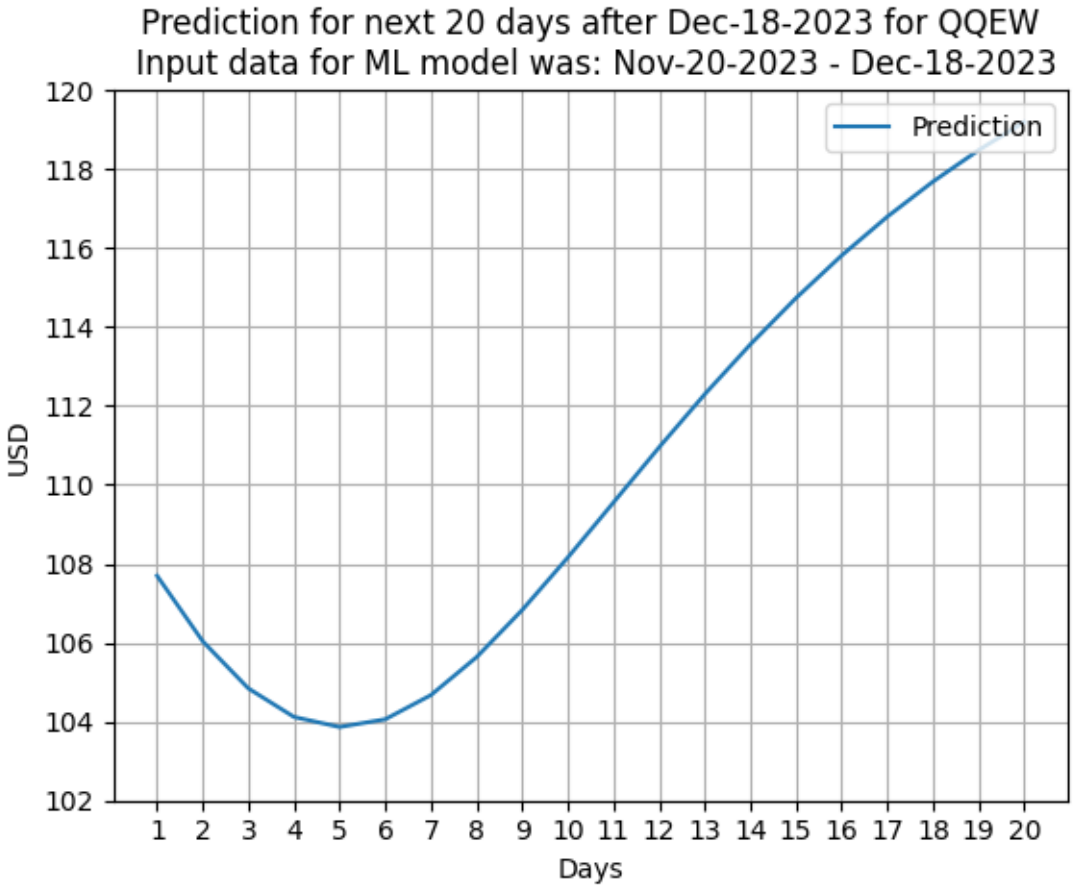
prepare date for axis labeling

[29]:
```
# TODO: Needs to implement calender
# dates = [ date.date().isoformat() for date in dataset['splits']['df_predict'].
 ↪index.tolist()]
# # dates
```

[30]:
```
# print(inv_true_forecast.shape)
# print(inv_pred_forecast.shape)
```

[31]:
```
plt.plot(inv_pred_forecast, label='Prediction')
plt.xlabel('Days')
plt.ylabel('USD')
leg = plt.legend(loc='upper right')
plt.yticks(ticks=plt.yticks()[0], labels=plt.yticks()[0].astype(int))
# plt.xticks(ticks=list(range(20)), labels=dates, rotation = 90) # TODO: Needs␣
 ↪to implement calender
plt.xticks(np.arange(len(inv_pred_forecast)), np.arange(1,␣
 ↪len(inv_pred_forecast)+1))
```

```
# plt.title("Prediction for QQEW (Oct 20, 2023 - Nov 16, 2023)")
plt.grid()
plt.title('Prediction for next 20 days after {} for QQEW\n Input data for ML␣
  ↪model was: {} - {}'.format(last_date, first_date, last_date))
plt.show()
```



Will be updated soon