0.1.ModelTrainingWalk-through

December 20, 2023

This notebook demonstrate full walk-through from data retrieval to model training

0.0.1 Import modules

[ ]:
```
from src.loader import DataLoader
from src.preprocessor import Preprocessor
from src.dataprep import DataPrepTraining
from src.model import CustomModel
```

0.0.2 Set up

[ ]:
```
ticker = 'QQEW'

# Required for Preprocessor and DataPrepTraining
data_version = 0

# Required for DataPrepTraining
test_size = 0.3

# Required for DataPrepTraining and CustomModel
rolling_window = 20
forecast_horizon = 20
```

0.0.3 Instantiate DataLoader class

[ ]:
```
data = DataLoader(ticker)
```

We can read data from yahoo finance using *read_remote* method. > Ticker already set during instantiation

TODO: Needs to implement calender and class to method for weekly processing

[ ]:
```
data.read_remote(until='2023-12-19', since='2006-5-2')
```

Retrieved data has been set as DataLoader attribute. Let's access data.

[ ]:
```
data.df.head(5)
```

We can write data to disk. Please set version to avoid ambiguity.

```
[ ]: data.save_raw_data(version=data_version)
```

```
[ ]:
```

We can instantiate read raw data from local storage as well.

```
[ ]: data = DataLoader(ticker)
     data.read_local(filepath='data/{}_RAW_V{}.csv'.format(ticker, data_version),␣
      ↪isRawData = True)
```

0.0.4 Instantiate Preprocessor class

Preprocessor is meant to perform technical analysis. It doesn't process data for training.

```
[ ]: prep = Preprocessor()
     prep.set_df(df=data.df, isRawData=True) # the next step will not run if␣
      ↪isRaWData=False
```

```
[ ]: ta = prep.prepare_technical()
```

Check the first 5 rows. First few rows will contain NaN due to the type of technical analysis.

```
[ ]: ta.head(5)
```

Check the last 5 rows. It should not contain any NaN

```
[ ]: ta.tail(5)
```

Use *save_technical_analysis_data* method from *DataLoader* class to export technical analysis

```
[ ]: data.save_technical_analysis_data(
         df=ta,
         ticker=ticker,
         version=data_version
     )
```

Technical data can be read using *read_local* method from *DataLoader* class.

```
[ ]: data = DataLoader(ticker)
     data.read_local(filepath='data/{}_TA_V{}.csv'.format(ticker, data_version),␣
      ↪isRawData = False) # TA = technocal analysis
```

$isRawData = False$ is required to read technical analysis data properly. Set $isRawData = True$ to read raw data as shown previously.

Check if it works

```
[ ]: data.df.tail(5)
```

0.0.5 Instantiate DataPrepTraining class

Use set_df method to set technical analysis data as class attribute

[ ]:
```
dataprep = DataPrepTraining()
dataprep.set_df(df=data.df)
```

Drop NaN and set rolling window, forecast horizon, and test size

[ ]:
```
dataprep.dropna()
dataprep.set_rw_fh_test_size(rw=rolling_window, fh=forecast_horizon,␣
 ↪test_size=test_size)
```

Split the dataset

[ ]:
```
splits = dataprep.generate_train_test_predict_split()
```

Let's see the splits

[ ]:
```
# TODO
# Implement validation set in splitter
print('Training set: last 5 rows')
print(splits['df_train'].tail(5))

print()
print('Test set: last 5 rows')
print(splits['df_test'].tail(5))

print()
print('Prediction set: last 5 rows')
print(splits['df_predict'].tail(5))
```

Normalise training set and test set. > Prediction set normalisation has not been implemented. > Normalisation will return dict that includes normalised_data_py_list and scalers. > rolling_window related operation done here

[ ]:
```
# Normalise all dataframes except prediction
normalised_train = dataprep.normalise_dataframe(df=splits['df_train'], step=1,␣
 ↪standard_norm=True)
normalised_test = dataprep.normalise_dataframe(df=splits['df_test'], step=1,␣
 ↪standard_norm=True)
```

Let's prepare features and labels. > Returns a dict containing features, labels, np.array(normalised_data_py_list)

[ ]:
```
training_set = dataprep.
 ↪prepare_feature_and_label(data_list=normalised_train['normalised_data_py_list'])
test_set = dataprep.
 ↪prepare_feature_and_label(data_list=normalised_test['normalised_data_py_list'])
```

0.0.6 Let's save everything!

[ ]:
```
data.save_all_data_for_model_training(
    datadict={
        'splits': splits,
        'normalised_train': normalised_train,
        'normalised_test':normalised_test,
        'training_set': training_set,
        'test_set': test_set
    },
    ticker=ticker,
    version=data_version)
```

1 Train model

[ ]:
```
model = CustomModel()
```

[ ]:
```
model.read_all_data_for_model_training(filepath='data/{}_PREPROCESSED_V{}.
↪pickle'.format(ticker, data_version))
```

[ ]:
```
X_train = model.dataset['training_set']['features']
y_train = model.dataset['training_set']['labels']
X_test = model.dataset['test_set']['features']
y_test = model.dataset['test_set']['labels']
```

Complile the model

[ ]:
```
model.build_model(
    input_n=int(rolling_window),
    output_n = int(forecast_horizon),
    drop_rate = 0.1, # this is equivalent to regularization
    latent_n = 400,
    feature_n = X_train.shape[2]
)
```

Train model

[ ]:
```
model.train(
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    epochs=60,
    batch_size=20,
    modelpath='model/QQEW_LSTM_RW20_FH20_V{}.h5'.format(data_version)
)
```

Model can be loaded using *read_model_local*() method of *ModelLoader* class