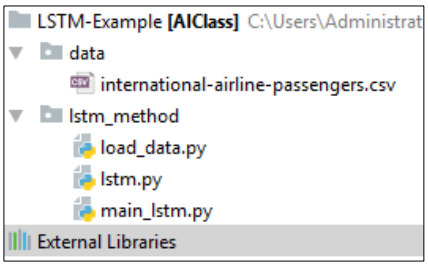
Predicting Airline Passenger Using LSTM

Create the directory hierarchy same as follow:



A. Save the following code to load_data.py

1. Importing some packages.

Pandas as data manipulator, MinMaxScaler for normalizing the data to particular range value, and Numpy for metric operation.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from pandas import concat
import numpy as np
```

2. Loading data from csv file.

The csv data consists of two columns, namely Month and Num of Passenger. set the column Month as date index which corresponding the time series data

```
def load_data(file):
    df = pd.read_csv(file,
                     index_col=["Month"],
                     usecols=["Month","Num of Passenger"])

    df.index = pd.to_datetime(df.index)
    return df
```

3. Converting series to supervised

Converting data to X and Y format, where the input is one-dimensional data it will create the Y data from X data. Such as follow:

X	Y
112	118
118	132
132	129

129	121
121	135

n_in corresponds to time steps before, and *n_out* corresponds to number of time steps after.

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j + 1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j + 1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j + 1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

4. Normalizing the data

The number of passenger value is so high and volatile, therefore to make it more tractable to predict we convert it into 0 to 1 range.

```
def normalize_data(data):
    scaler = MinMaxScaler(feature_range=(0, 1))
    data_scaled = scaler.fit_transform(data)
    return scaler, data_scaled
```

5. Inverting normalized data

This function for inverting normalize data; e.g. range 0 to 1, to the real value.

```
def inverse_normdata(scaler, dataX, yhat):
    dataX = dataX.reshape((dataX.shape[0], dataX.shape[2]))
    inv_y = np.concatenate((dataX, yhat), axis=1)
    inv_y = scaler.inverse_transform(inv_y)
    inv_y = inv_y[:, -1]
    return inv_y
```

B. Save the following code as lstm.py

1. Importing some packages.

The important package is Keras for creating LSTM model

```
import os
import time
import warnings
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # Hide messy TensorFlow warnings
warnings.filterwarnings("ignore")  # Hide messy Numpy warnings
```

2. Create LSTM model

Function for build LSTM model, where the input is array of layer which containing five index. layers [0,1,2,3,4,5] layer 0 correspond to dimension of data value, layer 1 correspond to the time-step, layer 2,3, and 4 corresponds to number of neurons in hidden layer and layer 5 correspond to the output.

```
#layers [0,1,2,3,4,5]
def build_model(layers):
    model = Sequential()

    model.add(LSTM(100,
        input_shape=(layers[1], layers[0]),
        return_sequences=True))
    model.add(Dropout(0.5))

    model.add(LSTM(
        layers[2],
        return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
        layers[3],
        return_sequences=True))
    model.add(Dropout(0.1))

    model.add(LSTM(
        layers[4],
        return_sequences=False))
    model.add(Dropout(0.5))

    model.add(Dense(
        output_dim=layers[5]))
    model.add(Activation("tanh"))

    start = time.time()
    model.compile(loss="mse", optimizer="adam")
    print("> Compilation Time : ", time.time() - start)
```

```
        model.reset_states()
        return model
```

C. Save the following code as main_lstm.py

1. Importing some packages

```
import lstm_method.load_data as data
import lstm_method.lstm as lstm
import time
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import mean_squared_error
from math import sqrt
```

2. Main class

```
if __name__=='__main__':
    global_start_time = time.time()
    epochs = 100
    df = data.load_data("../data/international-airline-passengers.csv")

    data_reframed = data.series_to_supervised(df, 1, 1)

    df_data_reframed = pd.DataFrame(data_reframed)

    scaler, data_scaled = data.normalize_data(data_reframed.values)
    df_data_scaled = pd.DataFrame(data_scaled, index=df_data_reframed.index,
columns=df_data_reframed.columns)

    print("data scaled shape = ", df_data_scaled.shape)
    n_train_days = df_data_scaled.shape[0] - 12
    train = df_data_scaled.iloc[:n_train_days, :]
    test = df_data_scaled.iloc[n_train_days:, :]

    print("train shape = ", train.shape)
    print("test shape = ", test.shape)

    train_X, train_y = train.iloc[:, :-1], train.iloc[:, -1]
    test_X, test_y = test.iloc[:, :-1], test.iloc[:, -1]

    # reshape input to be 3D [samples, timesteps, features]
    train_X = train_X.values.reshape((train_X.shape[0], 1, train_X.shape[1]))
    test_X = test_X.values.reshape((test_X.shape[0], 1, test_X.shape[1]))

    train_y = pd.DataFrame(train_y)
    test_y = pd.DataFrame(test_y)

    print("train_X shape = %s, train_y shape = %s, test_X shape = %s, test_y shape = %s" %
(train_X.shape, train_y.shape, test_X.shape, test_y.shape))

    model = lstm.build_model([train_X.shape[2], train_X.shape[1], 255, 725, 755, 1])
```

```
    history = model.fit(train_X, train_y, epochs=epochs, batch_size=len(train_X),
validation_data=(test_X, test_y),
                        verbose=2, shuffle=False)
    # plot history

    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='test')
    plt.legend()
    plt.show()

    # plot model and train data
    y_train = model.predict(train_X)
    inv_ymodel = data.inverse_normdata(scaler, train_X, y_train)
    df_inv_ymodel = pd.DataFrame(inv_ymodel, index=train_y.index)

    # plot test data
    y_test = model.predict(test_X)
    inv_y_test = data.inverse_normdata(scaler, test_X, y_test)

    df_inv_ytest = pd.DataFrame(inv_y_test, index=test_y.index)
    plt.plot(df, color='orange', label='Actual')
    plt.plot(df_inv_ymodel, color='green', label='Trained')
    plt.plot(df_inv_ytest, color='red', label='Predicted')
    plt.legend()
    plt.show()

    inv_test_y = data.inverse_normdata(scaler, test_X, test_y)
    rmse_normal = sqrt(mean_squared_error(inv_test_y, inv_y_test))
    rmse_scaled = sqrt(mean_squared_error(test_y, y_test))
    print('Test RMSE normal : %.3f' % rmse_normal)
    print('Test RMSE scaled : %.3f' % rmse_scaled)
```

```
 - 0s - loss: 0.0037 - val_loss: 0.0128
Epoch 97/100
 - 0s - loss: 0.0047 - val_loss: 0.0127
Epoch 98/100
 - 0s - loss: 0.0044 - val_loss: 0.0127
Epoch 99/100
 - 0s - loss: 0.0038 - val_loss: 0.0127
Epoch 100/100
 - 0s - loss: 0.0047 - val_loss: 0.0125
Test RMSE normal : 57.919
Test RMSE scaled : 0.112

Process finished with exit code 0
```

Figure 1. Epoch and RMSE

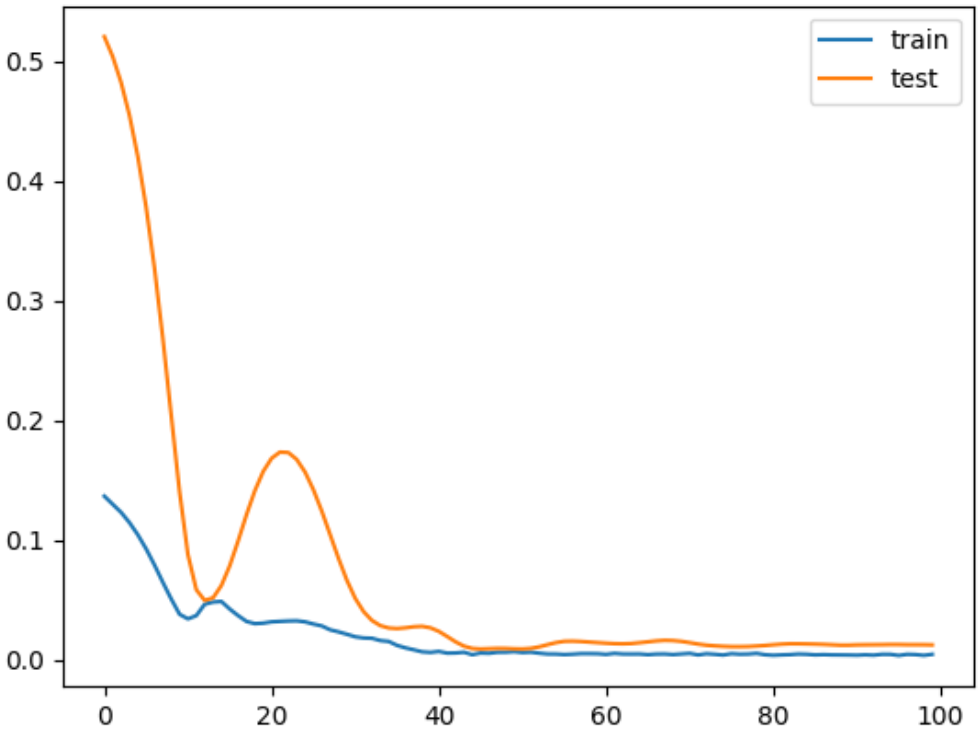


Figure 2. Loss function

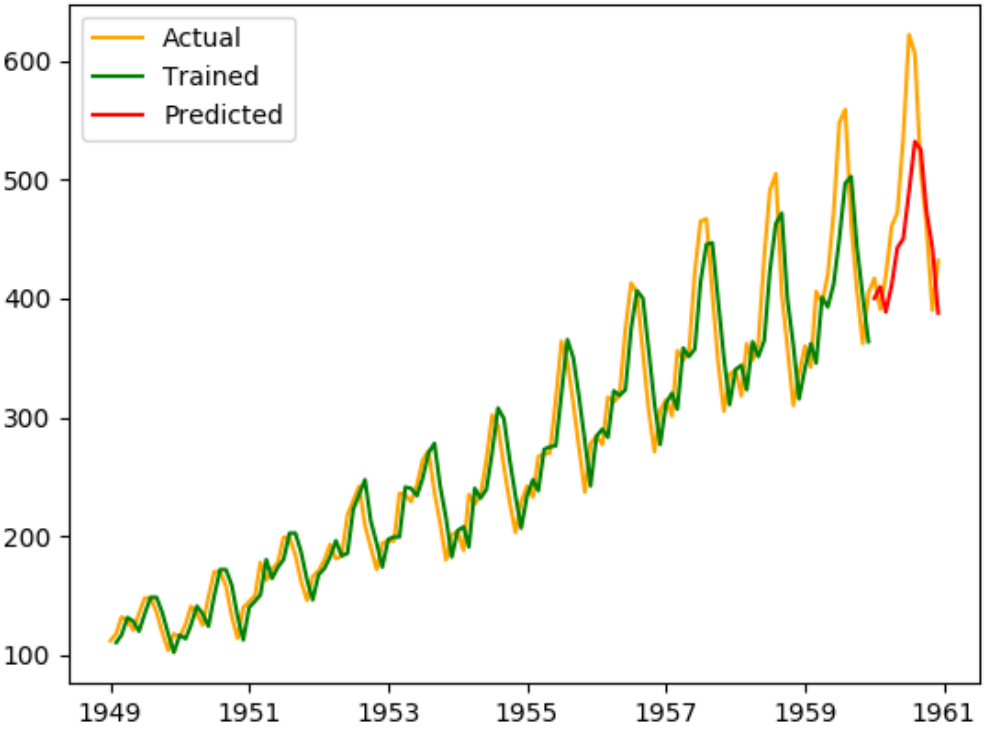


Figure 3. trained and predicted data