TIME SERIES ANALYSIS ELECTRIC POWER CONSUMPTION PREDICTION

FEBRUARY 17, 2023

MUHAMMAD AMIN GHIAS

Table of Contents

ABSTRACT:

Electricity is now a necessity of life, knowing how much power will be consumed in future is beneficial for a city. This project will consider previous observations of power and other features to predict power consumption in future. We will use RNN and LSTM deep learning time series to make predictions.

INTRODUCTION

The demand of electrical power keeps on increasing in a city and predicting future power required can help the city plan ahead to meet the power requirements. RNN-LSTM will be used to make predictions

MAIN OBJECTIVE:

The main objective of this project is to predict the power consumption of a city using RNN-LSTM of the time series data.

DATA ACQUISITION AND WRANGLING

DATA SOURCE

For this project we will use online data the from UCL machine learning repository

CSV files are from url:

https://archive.ics.uci.edu/ml/datasets/Power+consumption+of+Tetouan+city

The Dataset consists of 8 columns and 52416 rows.

DATA DESCRIPTION

The description of columns is:

- Date Time: Each ten minutes.
- Temperature: Weather Temperature of Tetouan city.
- Humidity: Weather Humidity of Tetouan city.
- Wind Speed of Tetouan city.
- general diffuse flows
- diffuse flows
- power consumption of zone 1 of Tetouan city.
- power consumption of zone 2 of Tetouan city.
- power consumption of zone 3 of Tetouan city.

```python

df=pd.read_csv('Tetuan City power consumption.csv',index_col=0, parse_dates=True, squeeze=True)
print(df.shape)
df.head()
```

✓ 16.6s Python

```
(52416, 8)
```

	Temperature	Humidity	Wind Speed	general diffuse flows	diffuse flows	Zone 1 Power Consumption	Zone 2 Power Consumption	Zone 3 Power Consumption
DateTime								
2017-01-01 00:00:00	6.559	73.8	0.083	0.051	0.119	34055.69620	16128.87538	20240.96386
2017-01-01 00:10:00	6.414	74.5	0.083	0.070	0.085	29814.68354	19375.07599	20131.08434
2017-01-01 00:20:00	6.313	74.5	0.080	0.062	0.100	29128.10127	19006.68693	19668.43373
2017-01-01 00:30:00	6.121	75.0	0.083	0.091	0.096	28228.86076	18361.09422	18899.27711
2017-01-01 00:40:00	5.921	75.7	0.081	0.048	0.085	27335.69620	17872.34043	18442.40964

DATA CLEANING & Preparation

Resampling data to 1 hour time frame and taking mean of the feature for that 1 hour

```python
hourly = df.resample('H').mean()

# dropping power consuption columns
hourly=hourly.drop(columns=['Zone 1 Power Consumption',
        'Zone 2  Power Consumption', 'Zone 3  Power Consumption'])
print(hourly.shape)
hourly.head()
```

✓ 0.2s

```
(8736, 5)
```

	Temperature	Humidity	Wind Speed	general diffuse flows	diffuse flows
DateTime					
2017-01-01 00:00:00	6.196833	75.066667	0.081833	0.063500	0.098833
2017-01-01 01:00:00	5.548833	77.583333	0.082000	0.056833	0.112500
2017-01-01 02:00:00	5.054333	78.933333	0.082333	0.063000	0.129167
2017-01-01 03:00:00	5.004333	77.083333	0.082833	0.059833	0.141000
2017-01-01 04:00:00	5.097667	74.050000	0.082333	0.058000	0.122833

making a dataset for power consumption and resampling them to 1 hour based on sum

```python
hourly_pc=df[['Zone 1 Power Consumption',
        'Zone 2  Power Consumption', 'Zone 3  Power Consumption']].resample('H').sum()

print(hourly_pc.shape)
hourly_pc.head()
```

✓ 0.2s

```
(8736, 3)
```

DateTime	Zone 1 Power Consumption	Zone 2 Power Consumption	Zone 3 Power Consumption
2017-01-01 00:00:00	175187.84810	108160.48632	115512.28916
2017-01-01 01:00:00	147943.29114	96470.51672	102257.34940
2017-01-01 02:00:00	132498.22784	85984.19453	94056.86747
2017-01-01 03:00:00	124866.83544	79316.71732	89303.13253
2017-01-01 04:00:00	122855.69620	77529.48328	85902.65060

Combining the two 1 hour sampled datasets

Combining the two 1 hour sampled datasets

```python
df_1hr=hourly.copy()

df_1hr['Zone 1 Power Consumption'] = hourly_pc['Zone 1 Power Consumption']

df_1hr.head()
```

✓ 0.1s

DateTime	Temperature	Humidity	Wind Speed	general diffuse flows	diffuse flows	Zone 1 Power Consumption
2017-01-01 00:00:00	6.196833	75.066667	0.081833	0.063500	0.098833	175187.84810
2017-01-01 01:00:00	5.548833	77.583333	0.082000	0.056833	0.112500	147943.29114
2017-01-01 02:00:00	5.054333	78.933333	0.082333	0.063000	0.129167	132498.22784
2017-01-01 03:00:00	5.004333	77.083333	0.082833	0.059833	0.141000	124866.83544
2017-01-01 04:00:00	5.097667	74.050000	0.082333	0.058000	0.122833	122855.69620

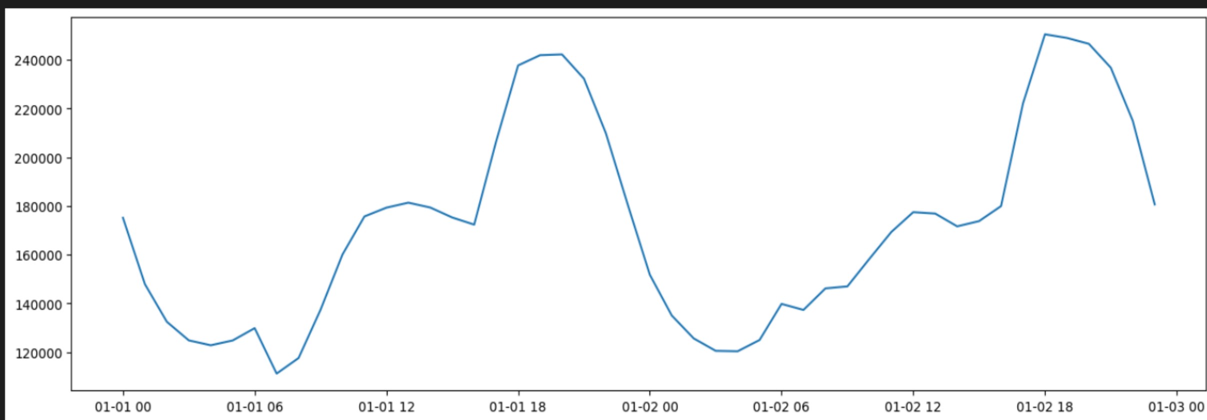
Trend

trend of first 48 hours

```python
plt.figure(figsize = (16,5))
plt.plot(df_1hr['Zone 1 Power Consumption'][0:(48)])
```

✓ 0.7s Python

```
[<matplotlib.lines.Line2D at 0x13d064fcd90>]
```



In this dataste we will use the data of all 5 features ['Temperature', 'Humidity', 'Wind Speed', 'general diffuse flows','diffuse flows'] for last 6 hours (t-6) and predict the power consumption current at (t=0)

Making a sequence for the model to learn and make predictions

```python
def making_sequence(data,input_timesteps,output_at_time=1,predicting_coulmn_names='Zone 1 Power Consumption'):

    data_list=[]
    df_s=data


    # for input sequence (t-3,t-2,t-1)

    for i in range(input_timesteps, 0, -1):
    # for i in range(last_hours):
        print(i)
        data_list.append(df_s.shift(i))

    # for output sequence (t,t+1)
    # for predicting single output
    for i in range(output_at_time):
        print(i)
        data_list.append(df_s[predicting_coulmn_names].shift(-i))

    df_s=pd.concat(data_list,axis=1)

    print(df_s.shape)
    # df_s.head()
    df_s=df_s.dropna()
    print(df_s.shape)
    df_s.head()
    return(df_s)
```

Scaling and transformation the dataset sequence

```python
# Scaling and transformation the dataset sequence
```

```python
def scaling(data):
    values=data.values
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(values)
    print(values.shape)
    return(scaled,scaler)
```

✓ 0.1s

Making train test datasets

```python

def train_test(data,n_features,n_hours,train_percent,output_at_time):


    train_percent =train_percent/100

    split_percent= round(train_percent * len(data))

    train = data[:split_percent, :]
    print('train',train.shape)
    test = data[split_percent:, :]
    print(test.shape)


    # split into input and outputs
    n_obs = n_hours * n_features
    trainX, trainy = train[:, :n_obs], train[:, -output_at_time]
    testX, testy = test[:, :n_obs], test[:, -output_at_time]
    print(trainX.shape, len(trainX), trainy.shape)
    # reshape input to be 3D [samples, timesteps, features]
    trainX = trainX.reshape((trainX.shape[0], n_hours, n_features))
    testX = testX.reshape((testX.shape[0], n_hours, n_features))
    print(trainX.shape, trainy.shape, testX.shape, testy.shape)
    return(trainX, trainy,testX, testy)
```

✓ 0.1s

Deep Learning Models for predictions

Making model for LSTM

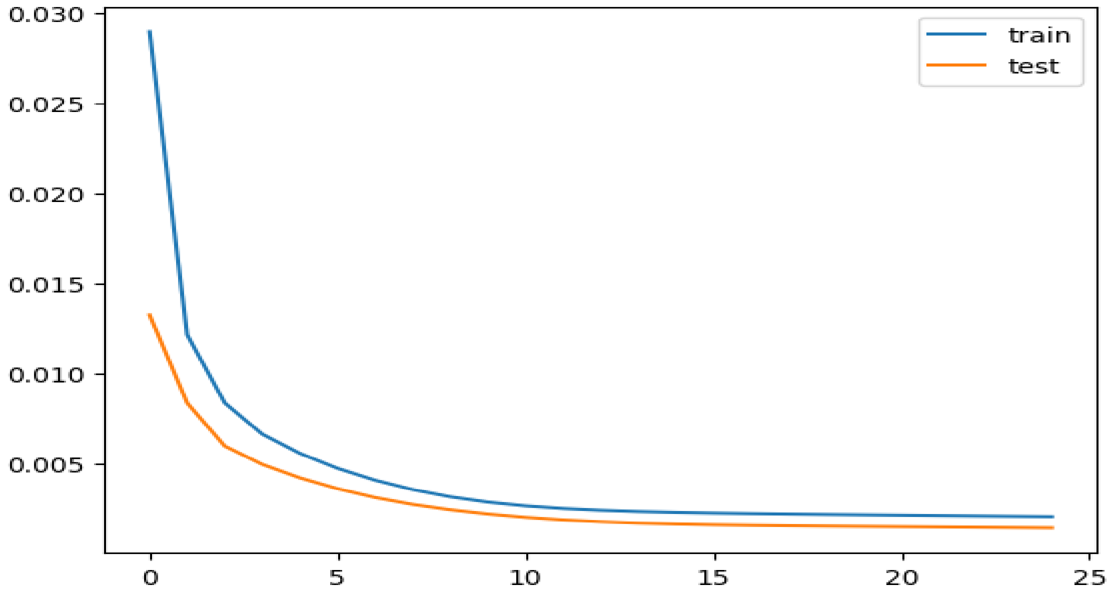
```python

def model_LSTM(batch_size,epoch, optimizer='adam',train_X=train_X, train_y=train_y, test_X=test_X, test_y=test_y):

    # design network
    model = Sequential()
    model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer=optimizer)

    model.summary()
    # fit network
    history = model.fit(train_X, train_y, epochs=epoch, batch_size=batch_size, 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()
    return(history,model)
```

Model of RNN

```python

def model_RNN(batch_size,epoch, optimizer='adam',train_X=train_X, train_y=train_y, test_X=test_X, test_y=test_y):

    # design network
    model = Sequential()
    model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2])))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer=optimizer)

    model.summary()
    # fit network
    history = model.fit(train_X, train_y, epochs=epoch, batch_size=batch_size, 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()
    return(history,model)
```

Applying LSTM model

The trend of model loss of training and test are

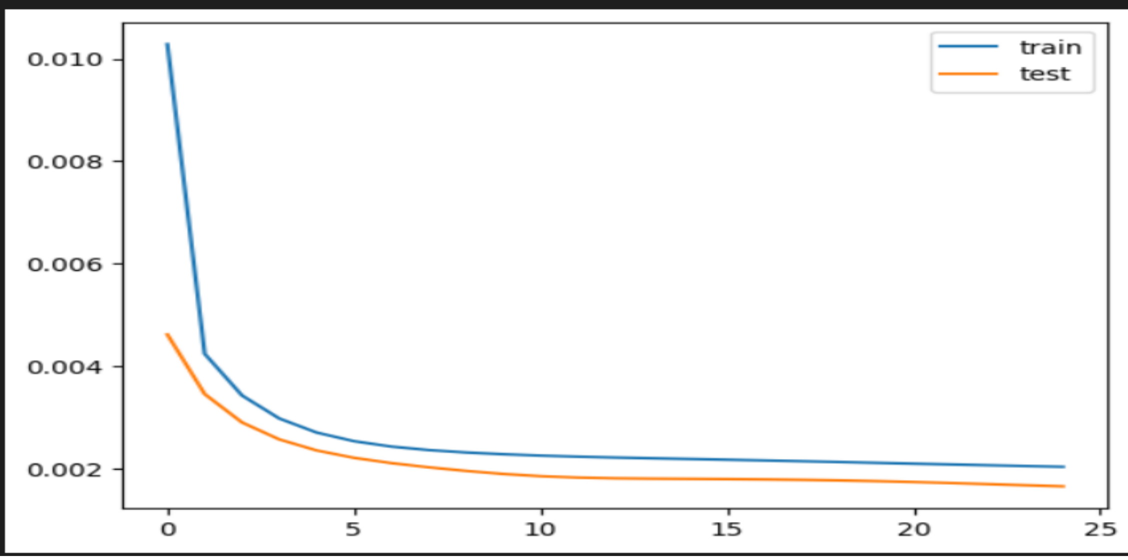



```
219/219 [==============================] - 3s 6ms/step
(6984, 1)
(6984, 36)
Train RMSE: 11984.812
Train R2: 0.921
55/55 [==============================] - 0s 6ms/step
(1746, 1)
(1746, 36)
Test RMSE: 8447.100
Test R2: 0.947
```

Applying RNN model

The trend of model loss of training and test are

```
Epoch 6/25
194/194 - 1s - loss: 0.0025 - val_loss: 0.0022 - 1s/epoch - 7ms/step
...
Epoch 24/25
194/194 - 1s - loss: 0.0021 - val_loss: 0.0017 - 1s/epoch - 6ms/step
Epoch 25/25
194/194 - 1s - loss: 0.0020 - val_loss: 0.0017 - 1s/epoch - 6ms/step
```




```
219/219 [==============================] - 1s 4ms/step
(6984, 1)
(6984, 36)
Train RMSE: 14778.392
Train R2: 0.880
55/55 [==============================] - 0s 4ms/step
(1746, 1)
(1746, 36)
Test RMSE: 9158.622
Test R2: 0.938
```

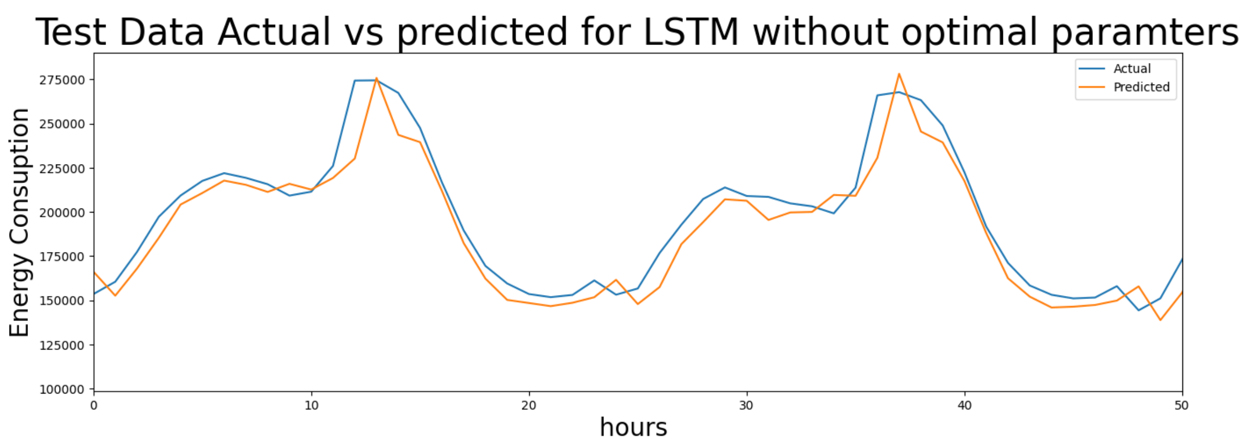
RESULTS:

✓ 0.1s

	model_name	train_rmse	test_rmse	train_r2	test_r2
0	LSTM-1	11984.812042	8447.099911	0.920977	0.946862
1	RNN	14778.392043	9158.622391	0.879844	0.937533

This shows that LSTM result is better than RNN

There prediction on test data can be plotted as shown in figure



LSTM prediction curve is close and better to actual.

DISCUSSION:

In this project we used RNN, LSTM for time series prediction of power consumption, the results are satisfactory

CONCLUSION

Using the LSTM RNN we were able to predict future power consumption, both models performed good but LSTM performed better compared to RNN

SUGGESTIONS

In future we can also use bidirectional LSTM to make predictions.