

Objective:

The goal is to forecast attribute x6 at location 6 for each of the following 20-time steps given a time series data consisting of 6 qualities observed daily over many years at different locations (days).

Data preprocessing and Methods:

The dataset contains 22310 labeled data with six numerical features (x0, x1, x2, x3, x4, x5, and x6) and one categorical feature location (values ranging from 0 to 8).

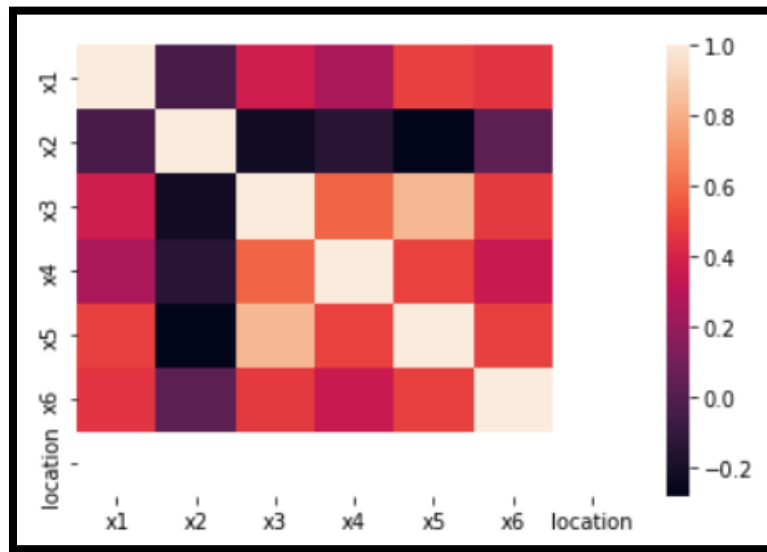


Fig 1: Correlation heatmap

For the data, the following assumptions are made:

- 1) A value of 0 is regarded as a NaN value.
- 2) Locations are not dependent on one another in a sequential order.
- 3) For prediction, only location 6 values are considered.

The data is first sorted by date, and then location 6 is used to filter it. As a result, our current dataset has been reduced to 2500 and the feature x2 is removed since the correlation graph shows that x6 is not dependent on x2.

The data is further separated into three sets: train, validation, and test, with a 70:20:10 ratio.

Because it works well with time series data, the interpolate function is used to substitute the NaN values.

Interpolating data based on dates would not be a suitable solution for this data because there is no information about location 6 available every day.

	x1	x3	x4	x5	x6
7394	0.203980	0.475610	0.333333	0.352941	0.067568
7393	0.293532	0.573171	0.333333	0.411765	0.073359
7392	0.363184	0.719512	0.424242	0.647059	0.117761
7391	0.582090	0.207317	0.272727	0.235294	0.063707
7390	0.303483	0.500000	0.515152	0.176471	0.055985
7389	0.208955	0.670732	1.000000	0.352941	0.084942
7388	0.363184	0.634146	0.333333	0.470588	0.088803
7387	0.393035	0.500000	0.363636	0.352941	0.065637
7386	0.333333	0.304878	0.333333	0.294118	0.081081
7385	0.328358	0.341463	0.333333	0.294118	0.073359

Fig 2: data after interpolation and scaler transform

Here there was an option to see whether there is dependency between locations. For that, 8 new columns were created which was a combination of x6 and location6.

This may also be done with the other five variables, although the model's complexity rises as more features are added. As a result, I just expanded the target variable. The resulted data from the above preprocessing is shown in the figure.

	x1	x2	x3	x4	x5	x6_0	x6_1	x6_2	x6_3	x6_4	x6_5	x6_6	x6_7	x6_8
0	43	6	37	11	10	0	0	0	0	0	0	0	0	66
1	41	9	39	11	6	0	0	0	0	0	0	0	0	70
2	0	0	9	4	5	0	0	0	0	0	0	0	0	64
3	0	0	16	3	6	0	0	0	0	0	0	0	0	90
4	59	5	53	10	13	0	0	0	0	0	0	0	0	86
5	0	0	2	2	2	0	0	0	0	0	0	0	0	42
6	0	0	31	5	9	0	0	0	0	0	0	0	0	46
7	0	0	26	10	9	0	0	0	0	0	0	0	0	70
8	0	0	50	6	8	0	0	0	0	0	0	0	0	48
9	59	9	47	11	7	0	0	0	0	0	0	0	0	76

Fig3: New features from the combination of x6 and location 6

The issue with the foregoing alternatives is that the dataset contains many zeroes, making it a sparse matrix. As a result, it will have no bearing on the forecast (it gives the same accuracy).

Because we're working with time series data, I used PCA to combine all six characteristics into a single feature before applying the ARIMA model. The model performed poorly because the single feature could not handle all the dataset's variation.

Then LSTM model was proposed for this task as the model works well for multivariate and forecasting problems.

For the LSTM model the below were the configurations used:

- Hidden size=40
- Number of Layers=1
- Number of features =5

- Output size=20(forecast 20 x6 values from T to T+19)
- Batch size=16

The length of the series should be bigger than the horizon value, thus a value of 20 or more was recommended.

As a result, the sequence length was set to 20 at the start.

For this model, MAE was used as the loss function.

The train loss was 0.02359, while the validation loss was 0.02025, according to the model.

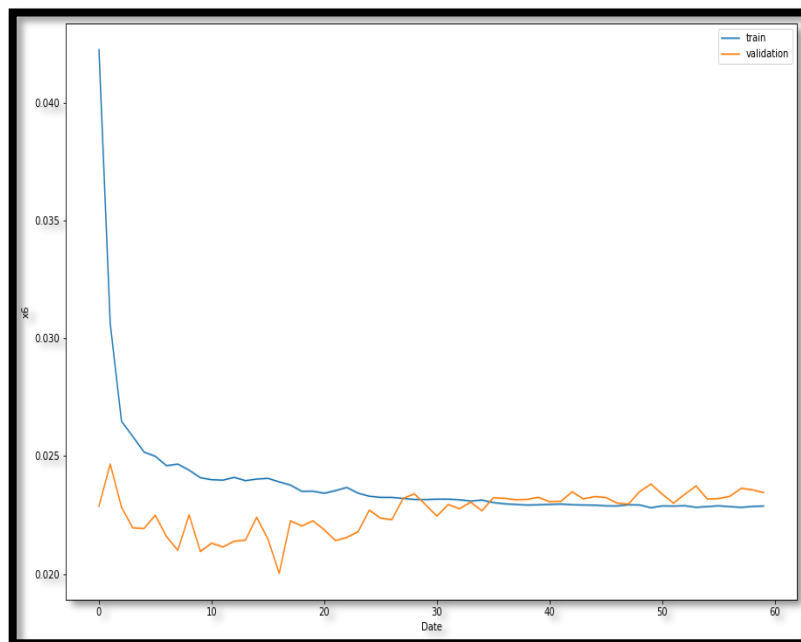


Fig4: Train and validation loss for the LSTM model.

The loss for scaled data with the test set was 0.0038. The entire date was fed into the model to test how closely it matched the real data, and the results were nearly identical.

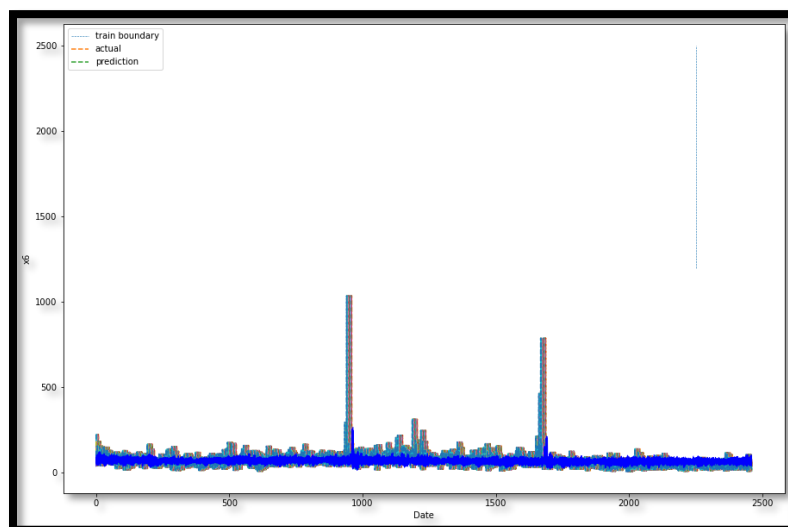


Fig5: actual and predicted data

The model was even trained on the 14 features of train data and showed no improvement over the prior results..

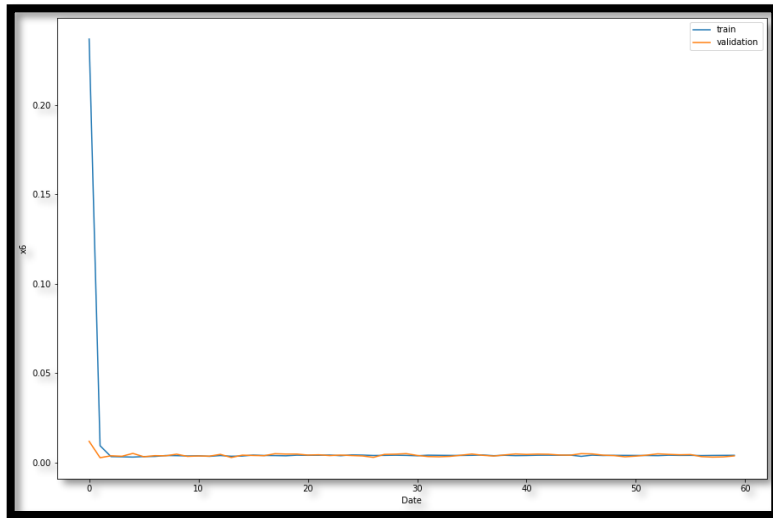


Fig6:Train and validation loss

The data was even trained with RNN and there wasn't any improvement with same configuration set.

Model and parameter selection

The following characteristics of the LSTM made it a good option for model choice over RNN:

- LSTM does not have a vanishing gradient problem.
- Other models, such as autoencoder and transformer, were not utilized since the data was sparse and the models' complexity was high.
- The size of the hidden layer is chosen 40 .
- A sequence length of 20 was a decent option because the data was restricted and the model needed to look back at earlier time steps up to T-horizon to anticipate the horizon.
- The epoch of 30 was chosen since the model did not improve significantly when the epoch was increased.
- The LSTM's hidden state and cell state were chosen at random.

Summary

With our final LSTM model with above optimal hyperparameters selected, we get following performance measures:

Dataset feature size:5

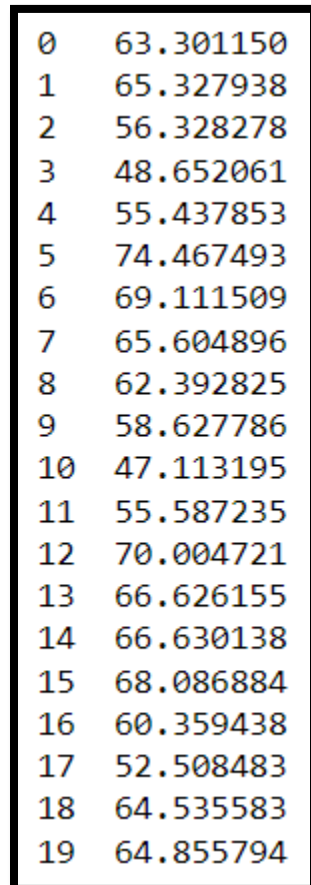
Train Loss: 0.02359

Val Loss: 0.02025

Test Loss (scalar data): 0.00389

Test Loss (Unscaler data): 25.12864108378266

The predicted forecast for x6 for next 20 days is as follows:



0	63.301150
1	65.327938
2	56.328278
3	48.652061
4	55.437853
5	74.467493
6	69.111509
7	65.604896
8	62.392825
9	58.627786
10	47.113195
11	55.587235
12	70.004721
13	66.626155
14	66.630138
15	68.086884
16	60.359438
17	52.508483
18	64.535583
19	64.855794

Fig 7: Prediction

Future Scope

Autoencoder can be used here to better insight of data and can even improve performance of the model.

Appendix

Here is the Python code of LSTM model: -

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
```

```

# Data Reading from existing csv file
df = pd.read_csv('./MTLprojectdata2022.csv')
df=df.sort_values(by="date")
df=df[df['location']==6]
pd.set_option('display.max_columns', None)
df.replace(0, np.nan, inplace=True)
df['x1']=df['x1'].interpolate().bfill().ffill()
df['x2']=df['x2'].interpolate().bfill().ffill()
df['x3']=df['x3'].interpolate().bfill().ffill()
df['x4']=df['x4'].interpolate().bfill().ffill()
df['x5']=df['x5'].interpolate().bfill().ffill()
df['x6']=df['x6'].interpolate().bfill().ffill()
#print("data_before_scaling", df)
# Data Scaling for training data
df_sorted=df
scaler = MinMaxScaler()
df_sorted[['x1', 'x3', 'x4','x5']] = scaler.fit_transform(df_sorted[['x1', 'x3', 'x4','x5']])
scaler2 = MinMaxScaler()

df_sorted['x6']= scaler2.fit_transform(df_sorted[['x6']])
# Data after scaling
print("data_after_scaling", df_sorted)

# Data Extract from 'Close Value'
X = df_sorted[['x1', 'x3', 'x4','x5','x6']].values
Y=df_sorted[['x6']].values
# device define to 'cpu'
#device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Used 300days data for the training, 50days for the validation and rest of days for test data
split1 = 1750
split2 = 2250
Horizon = 20

# 5days data used for prediction
sequence_length = 20

# Generating the sequence data of the features and target
def seq_data_for_multiple_feature(x,y,sequence_length):
    x_seq = []
    y_seq = []
    for i in range(len(x) - sequence_length - Horizon):
        x_seq.append(x[i: i + sequence_length])
        # y_seq.append(x[i + sequence_length + Horizon])
        y_seq.append(y[i+sequence_length:i + sequence_length + Horizon])
    #return torch.FloatTensor(x_seq).to(device), torch.FloatTensor(y_seq).to(device).view([-1, 1])
    return torch.FloatTensor(x_seq), torch.FloatTensor(y_seq)
x_seq, y_seq = seq_data_for_multiple_feature(X,Y,sequence_length)
#print(x_seq)

```

```

#print(x_seq.size())

# train , Validation & test data split (1 to 300 /300 to 350/ 350 to 430)
x_train_seq = x_seq[:split1]
y_train_seq = y_seq[:split1]
x_valid_seq = x_seq[split1:split2]
y_valid_seq = y_seq[split1:split2]
x_test_seq = x_seq[split2:]
y_test_seq = y_seq[split2:]

# train / validation / test sets defined
train = torch.utils.data.TensorDataset(x_train_seq, y_train_seq)
valid = torch.utils.data.TensorDataset(x_valid_seq, y_valid_seq)
test = torch.utils.data.TensorDataset(x_test_seq, y_test_seq)

# batch_size
batch_size = 16

# Generating batch sample from the train / validation / test sets
train_loader = torch.utils.data.DataLoader(dataset=train, batch_size=batch_size, shuffle=False)
valid_loader = torch.utils.data.DataLoader(dataset=valid, batch_size=batch_size, shuffle=False)
test_loader = torch.utils.data.DataLoader(dataset=test, batch_size=batch_size, shuffle=False)

# Define RNN structure
class LSTM_STRUCTURE(nn.Module):
    def __init__(self, input_size, hidden_size, sequence_length, num_layers):
        super(LSTM_STRUCTURE, self).__init__()
        #self.device = device
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.rnn = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Sequential(nn.Linear(hidden_size * sequence_length, 20), nn.LeakyReLU())

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size()[0], self.hidden_size)
        out, hn = self.lstm(x)
        out = out.reshape(out.shape[0], -1) # many to many strategy
        #print("Shape of output for many_to_many strategy",out.size())
        out = self.fc(out)
        #print("Shape of output for many_to_many strategy",out.size())
        return out

input_size = x_seq.size(2) # single value (close value of stock), it is the third dimension that means the number o
f sensors
num_layers = 1
hidden_size = 40 # both hidden layers of size 8

# Gnerating model for example 1
model_1 = LSTM_STRUCTURE(input_size=input_size,hidden_size=hidden_size,sequence_length=sequence_le
ngth,num_layers=num_layers)

```

```

# Loss functions for regression
criterion = nn.L1Loss()
# Learning rate
lr = 1e-3
# Number of epoch
num_epochs = 30
# Optimizer
optimizer = optim.Adam(model_1.parameters(), lr=lr)

# containers for stats
loss_stats = {
    "train": [],
    "val": []
}

def training(num_epochs, train_loader, valid_loader, criterion, optimizer, model):
    # Training part, see the loss by epoch
    for epoch in range(num_epochs):

        # Training
        train_epoch_loss = 0
        for X_train_batch, y_train_batch in train_loader:
            X_train_batch, y_train_batch = X_train_batch, y_train_batch
            optimizer.zero_grad()
            y_train_pred = model(X_train_batch)
            y_train_batch=y_train_batch.reshape([y_train_batch.shape[0],y_train_batch.shape[1]])
            train_loss = criterion(y_train_pred.reshape(y_train_pred.shape[0],y_train_pred.shape[1]), y_train_batch)
            #print("Training prediction y",y_train_pred)
            #print("Training batch y",y_train_batch)
            train_loss.backward()
            optimizer.step()
            train_epoch_loss += train_loss.item()

        # validation
        with torch.no_grad():
            val_epoch_loss = 0
            for X_val_batch, y_val_batch in valid_loader:
                X_val_batch, y_val_batch = X_val_batch, y_val_batch
                y_val_pred = model(X_val_batch)
                y_val_pred=y_val_pred.reshape([y_val_pred.shape[0],y_val_pred.shape[1]])
                y_val_batch=y_val_batch.reshape([y_val_batch.shape[0],y_val_batch.shape[1]])
                val_loss = criterion(y_val_pred, y_val_batch)
                val_epoch_loss += val_loss.item()

        loss_stats["train"].append(train_epoch_loss / len(train_loader))
        loss_stats["val"].append(val_epoch_loss / len(valid_loader))
        print(f'Epoch {epoch}: \ Train loss: {train_epoch_loss / len(train_loader):.5f} \ Val loss: {val_epoch_loss / len(valid_loader):.5f}')

    plt.figure(figsize=(15, 10))
    plt.plot(loss_stats["train"])

```



```

plt.plot(loss_stats["val"])
plt.xlabel('epochs')
plt.ylabel('AVG_running_loss')
plt.legend(['train', 'validation'])
plt.xlabel('Date')
plt.ylabel('x6')
plt.show()
# Prediction results for training and testing
def plotting(actual, model, x_seq, y_seq, file_name):
    with torch.no_grad():
        total = []
        total_data = torch.utils.data.TensorDataset(x_seq, y_seq)
        data_loader = torch.utils.data.DataLoader(dataset=total_data, batch_size=batch_size, shuffle=False)

        for data in data_loader:
            seq, target = data
            out = model(seq)
            total += out.cpu().numpy().tolist()

    reverse_total = scaler2.inverse_transform(total)
    reverse_actual = scaler2.inverse_transform(y_seq.reshape(y_seq.shape[0], y_seq.shape[1]))

    plt.figure(figsize=(15, 10))
    plt.plot(np.ones(100) * split2, np.linspace(1200, 2500, 100), '--', linewidth=0.6)
    plt.plot(reverse_actual, '--')
    plt.plot(reverse_total, 'b', linewidth=0.6)
    plt.legend(['train boundary', 'actual', 'prediction'])
    plt.xlabel('Date')
    plt.ylabel('x6')
    plt.show()

# compute mean absolute error for Test Set
mae_loss = nn.L1Loss()
output = mae_loss(torch.tensor(reverse_actual[split2:]), torch.tensor(reverse_total[split2:])).numpy()
print("Mean Absolute Error on Test set is ", output)
#df2=pd.DataFrame()
#last window values
with torch.no_grad():
    out=model(x_seq[-1:])
    print(out.shape)
reverse_future=scaler2.inverse_transform(out.cpu().numpy().tolist())
print(reverse_future)
df2=pd.DataFrame(reverse_future)
df2.to_csv(file_name, sep='\t', encoding='utf-8')

# Model_1 training & loss graph by epochs
training(num_epochs, train_loader, valid_loader, criterion, optimizer, model_1)
# Plot the prediction results
file_name="./TemporalLearningReport2022SurajJyothiUnni.csv"
plotting(list(df['x6'][sequence_length:len(X) - Horizon]), model_1, x_seq, y_seq, file_name)

```