ANN

December 4, 2024

[86]:
```
# import necessary libraries
import pandas as pd
import numpy as np
```

[74]:
```
data = pd.read_csv('df_merged_weekly.csv', index_col=0)
data['dt'] = pd.to_datetime(data['dt'])
data.head()
```

[74]:

	week	dt	Cases	temp	feels_like	pressure	humidity	precipitation
0	2022-1	2022-01-01	7.0	298.38	299.64	1013.0	87.0	3.14
1	2022-2	2022-01-08	1.0	302.23	308.88	1011.0	68.0	0.00
2	2022-3	2022-01-15	2.0	300.74	304.09	1013.0	58.0	0.00
3	2022-4	2022-01-22	3.0	302.42	309.89	1010.0	63.0	0.00
4	2022-5	2022-01-29	3.0	303.37	308.69	1010.0	64.0	0.42

[75]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 140 entries, 0 to 139
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   week           140 non-null    object
 1   dt             140 non-null    datetime64[ns]
 2   Cases          140 non-null    float64
 3   temp           140 non-null    float64
 4   feels_like     140 non-null    float64
 5   pressure       140 non-null    float64
 6   humidity       140 non-null    float64
 7   precipitation  140 non-null    float64
```

```
dtypes: datetime64[ns](1), float64(6), object(1)
memory usage: 9.8+ KB
```

1 Columns

column	description	type
week	week corresponding to the specific year	datetime
dt	specific date	datetime
Cases	Dengue Cases	integer
temp	Average temperature in the given week	numeric
feels_like	Average feels_life in the given week	numeric
pressure	Average pressure in the given week	numeric
humidity	Average humidity in the given week	numeric
precipitation	Total precipitation in the given week	numeric

1.1 Training data

Data used to be trained range from 2022 to 2023

1.2 Testing data

Data used to be tested is 2024 (also used for training)

[76]:
```
# Feature Engineering
from sklearn.preprocessing import MinMaxScaler
standard_scaler = MinMaxScaler()
features = ["temp", "humidity", "precipitation"]
X = data[features]
X = standard_scaler.fit_transform(X)
temp = X[:,0]
humidity = X[:,1]
precipitation = X[:,2]
data['temp'] = temp
data['precipitation'] = precipitation
data['humidity'] = humidity
```

[77]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import mean_absolute_error, mean_squared_error
class Model(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size, dropouts):
        super().__init__()
        self.layer1 = nn.Linear(input_size, hidden_sizes[0])
        self.layer2 = nn.Linear(hidden_sizes[0], hidden_sizes[1])
        self.layer3 = nn.Linear(hidden_sizes[1], hidden_sizes[2])
```

```
        self.output = nn.Linear(hidden_sizes[2], output_size)

        self.dropouts = nn.ModuleList([nn.Dropout(dropout) for dropout in␣
↪dropouts])
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.layer1(x))
        x = self.dropouts[0](x)
        x = self.relu(self.layer2(x))
        x = self.dropouts[1](x)
        x = self.relu(self.layer3(x))
        x = self.dropouts[2](x)
        x = self.output(x)
        return x

def train_ANN(data, features, target, date_col, n_ahead):
    # Select relevant columns
    rel_col = features + [target] + [date_col]
    data = data[rel_col]

    # Getting the first and last index for the year 2024
    first_2024 = data[data[date_col].dt.year == 2024].index[0]
    last_2024 = data[data[date_col].dt.year == 2024].index[-1]

    first_2024subn = first_2024 - n_ahead
    last_2024subn = last_2024 - n_ahead

    # Define lags
    env_lags = [1,2,3,4]  # 2-week lag for environment features
    cases_lag = range(1, 12)  # 1 to 11 week lags for target variable

    # Create lagged features for environment and target variables
    for lag in env_lags:
        for feature in features:
            data[f'{feature}_lag_{lag}'] = data[feature].shift(lag)

    for lag in cases_lag:
        data[f'{target}_lag_{lag}'] = data[target].shift(lag)

    # Remove any rows with missing values due to lagging
    data = data.dropna()

    # Initialize list for storing predictions
    predict_data = []

    # Loop through data to get n-week ahead prediction
```

```
    index_begin = first_2024subn
    index_end = last_2024subn

    while index_begin <= index_end:
        # Split data into training and testing sets
        train_data = data[data.index <= index_begin]
        test_data = data.iloc[data.index.get_loc(index_begin):data.index.
↪get_loc(index_begin) + n_ahead]

        X_train = train_data.drop(columns=[target, date_col])
        y_train = train_data[target]
        X_test = test_data.drop(columns=[target, date_col])
        y_test = test_data[target]

        # Convert to PyTorch tensors
        X_train = torch.tensor(X_train.values, dtype=torch.float32)
        y_train = torch.tensor(y_train.values, dtype=torch.float32)
        X_test = torch.tensor(X_test.values, dtype=torch.float32)
        y_test = torch.tensor(y_test.values, dtype=torch.float32)

        # Combine inputs and labels into a Dataset
        train_dataset = TensorDataset(X_train, y_train)
        train_loader = DataLoader(train_dataset, batch_size=32, shuffle=False)

        # Initialize the ANN Model
        input_size = X_train.shape[1]
        hidden_sizes = [48, 32, 19]
        dropouts = [0.3, 0.2, 0.1]
        output_size = 1
        model = Model(input_size, hidden_sizes, output_size, dropouts)

        # Define optimizer and loss function
        optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
        loss_fn = nn.MSELoss()

        # Train the model
        model.train()
        num_epochs = 1000
        for epoch in range(num_epochs):
            for batch_X, batch_y in train_loader:
                optimizer.zero_grad()

                # Get model predictions (assuming output size is n_ahead)
                predictions = model(batch_X)

                # Compute loss (MSE)
                loss = loss_fn(predictions.squeeze(), batch_y)
```

```
                # Backpropagation
                loss.backward()
                optimizer.step()
            #print(f'Epoch {epoch+1}/{num_epochs} has passed')

        #  Evaluate on test data
        model.eval()
        with torch.no_grad():
            predictions = model(X_test).squeeze().numpy()
        try:
            predictions = predictions[-1]
        except IndexError:
            pass
        predict_data.append(predictions)
        index_begin += 1

    # Calculate Mean Absolute Error
    actual_data = data[data[date_col].dt.year == 2024][target].values
    MAE = mean_absolute_error(actual_data, predict_data)
    MSE = mean_squared_error(actual_data, predict_data)
    return predict_data, MAE, MSE
```

[78]:
```
# Perform Initial Testing
target="Cases"
features=["temp","humidity","precipitation"]
date = 'dt'
prediction_1_week, MAE_1_week, MSE_1_week = train_ANN(data, features, target,␣
 ↪date, 1)
prediction_4_week, MAE_4_week, MSE_4_week= train_ANN(data, features, target,␣
 ↪date, 4)
```

```
/tmp/ipykernel_5158/3166879319.py:46: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data[f'{feature}_lag_{lag}'] = data[feature].shift(lag)
/tmp/ipykernel_5158/3166879319.py:46: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data[f'{feature}_lag_{lag}'] = data[feature].shift(lag)
/home/miniloda/Documents/GitHub/Math-198.1---Special-
```

```
Problem/venv/lib/python3.10/site-packages/torch/nn/modules/loss.py:608:
UserWarning: Using a target size (torch.Size([1])) that is different to the
input size (torch.Size([])). This will likely lead to incorrect results due to
broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
/tmp/ipykernel_5158/3166879319.py:46: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data[f'{feature}_lag_{lag}'] = data[feature].shift(lag)
/tmp/ipykernel_5158/3166879319.py:46: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data[f'{feature}_lag_{lag}'] = data[feature].shift(lag)
/home/miniloda/Documents/GitHub/Math-198.1---Special-
Problem/venv/lib/python3.10/site-packages/torch/nn/modules/loss.py:608:
UserWarning: Using a target size (torch.Size([1])) that is different to the
input size (torch.Size([])). This will likely lead to incorrect results due to
broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

[87]:
```
print(f'The Mean Absolute Error of the 1-Week Ahead Model is {MAE_1_week}')
print(f'The Mean Absolute Error of the 4-Week Ahead Model is {MAE_4_week}')
print(f'The Root Mean Squared Error of the 1-Week Ahead Model is {np.
 ↪sqrt(MSE_1_week)}')
print(f'The Root Mean Squared Error of the 4-Week Ahead Model is {np.
 ↪sqrt(MSE_4_week)}')
```

```
The Mean Absolute Error of the 1-Week Ahead Model is 12.189200864897835
The Mean Absolute Error of the 4-Week Ahead Model is 17.1717133919398
The Root Mean Squared Error of the 1-Week Ahead Model is 23.6536483968945
The Root Mean Squared Error of the 4-Week Ahead Model is 34.12524544437012
```

[80]:
```
prediction_1_week[:5]
```

[80]:
```
[array(14.844003, dtype=float32),
 array(13.386841, dtype=float32),
 array(12.713357, dtype=float32),
 array(11.87863, dtype=float32),
 array(15.817395, dtype=float32)]
```

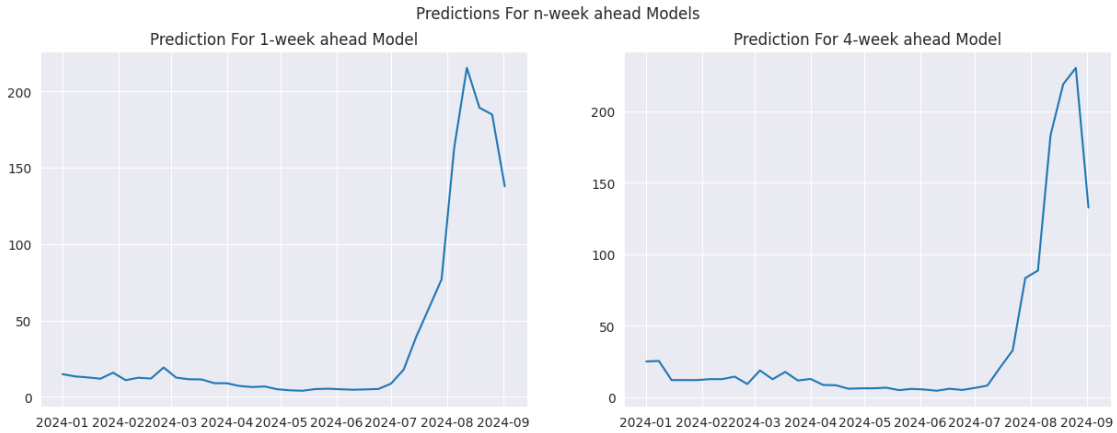
[81]:
```
prediction_4_week[:5]
```

[81]:
```
[np.float32(25.019201),
 np.float32(25.40394),
 np.float32(11.99758),
 np.float32(12.008745),
 np.float32(11.9914)]
```

2 Visualization

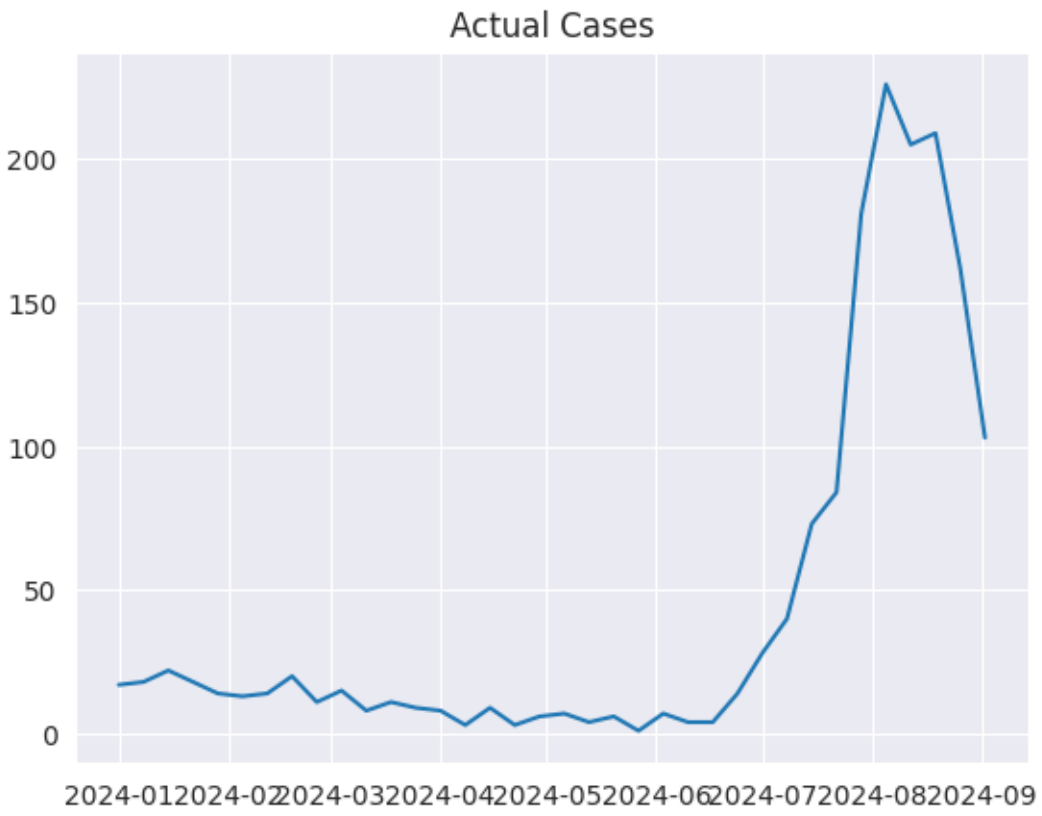
[82]:
```
import matplotlib.pyplot as plt
first_2024 = int((data['dt'].dt.year==2024).idxmax())
last_2024 = int(data.loc[data['dt'].dt.year == 2024].index[-1])
date = data.loc[first_2024:last_2024]['dt']
```

[83]:
```
fig, ax = plt.subplots(1,2, figsize=(15,5))
ax[0].plot(date,prediction_1_week)
ax[0].set_title('Prediction For 1-week ahead Model')
ax[1].plot(date,prediction_4_week)
ax[1].set_title('Prediction For 4-week ahead Model')
plt.suptitle('Predictions For n-week ahead Models')
plt.show()
```

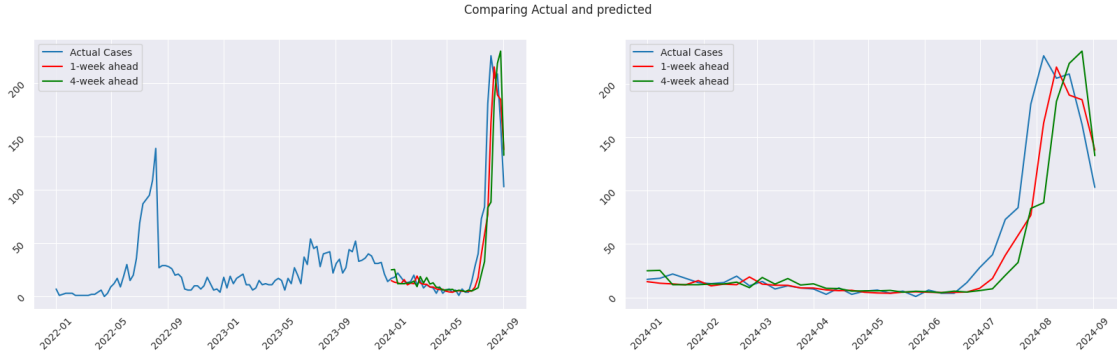


[84]:
```
actual_2024 = data[data['dt'].dt.year == 2024]
plt.plot(actual_2024['dt'], actual_2024['Cases'])
plt.plot()
plt.title('Actual Cases')
```

[84]:
```
Text(0.5, 1.0, 'Actual Cases')
```



[88]:
```
fig, ax = plt.subplots(1,2, figsize=(20,5))
ax[0].plot(data['dt'], data['Cases'], label = 'Actual Cases')
ax[0].plot(date, prediction_1_week, color = 'red', label = '1-week ahead')
ax[0].plot(date, prediction_4_week, color = 'green', label = '4-week ahead')
ax[0].tick_params(labelrotation=45)
ax[0].legend()
ax[1].plot(actual_2024['dt'], actual_2024['Cases'], label = 'Actual Cases')
ax[1].plot(date, prediction_1_week, color = 'red', label = '1-week ahead')
ax[1].plot(date, prediction_4_week, color = 'green', label = '4-week ahead')
ax[1].tick_params(labelrotation=45)
ax[1].legend()
plt.suptitle('Comparing Actual and predicted')
plt.show()
```


Comparing Actual and predicted
Actual Cases
1-week ahead
4-week ahead
0
50
100
150
200
2022-01
2022-05
2022-09
2023-01
2023-05
2023-09
2024-01
2024-05
2024-09
Actual Cases
1-week ahead
4-week ahead
0
50
100
150
200
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
2024-09

[ ]: