Shaastra Techathon - AI/ML Challenge 2024

The AI Challenge for the future

Team Name : AI SPIRITS
Team Lead : Kaushika A
Team Members : Pradeesh S and Yamunaasri T S

Description

This competition is about predicting the number of deaths in a widespread pandemic.

The Healthcare system in India was heavily overburdened during the peak of the pandemic Covid-19 three years ago. Hospital beds for emergency patients were in demand. Resources fell short of what was required. Being able to predict the number of cases can help draft a plan of action well in advance. Resources can be acquired and sent to people in need.

The dataset corresponds to a record of COVID-19 cases in India in various states from January 2020 to August 2021. It also contains the corresponding number of cured patients and deaths. Use your inner AI spirit to predict the number of deaths on a future date.

Dataset Preprocessing

Using pandas we checked the unique states and removed the unwanted states, we noticed that a few states were in different spelling so we replaced them with the original name to make sure there were 28 + 8 unique values in the states feature

```
data = data.drop(data[data["State/UnionTerritory"] == "Unassigned"].index)
data = data.drop(data[data["State/UnionTerritory"] == "Cases being reassigned to states"].index)

data = data.replace(to_replace = "Daman & Diu", value = "Dadra and Nagar Haveli and Daman and Diu")
data = data.replace(to_replace = "Telengana", value = "Telangana")

len(data["State/UnionTerritory"].unique())
# 28 + 8
```

```
36
```

Fig1. Training data preprocessing

```
test_data = test_data.replace(to_replace = "Bihar****", value = "Bihar")
test_data = test_data.replace(to_replace = "Madhya Pradesh***", value = "Madhya Pradesh")
test_data = test_data.replace(to_replace = "Maharashtra***", value = "Maharashtra")
test_data = test_data.replace(to_replace = "Dadra and Nagar Haveli", value = "Dadra and Nagar Haveli and Daman and Diu")
test_data = test_data.replace(to_replace = "Karanataka", value = "Karnataka")
test_data = test_data.replace(to_replace = "Himanchal Pradesh", value = "Himachal Pradesh")

len(test_data["State/UnionTerritory"].unique())
```

```
36
```

Fig2. Testing data preprocessing

Then we normalized the values using the MinMaxScaler library. Later we realised that each state has different ranges so it is optimal to normalize the data individually for each state. The train and test dataset is merged before normalizing.

Dataset visualization

To understand the given dataset we plotted the given features in a graph for each state individually. Each state has a unique range of deaths and patterns.

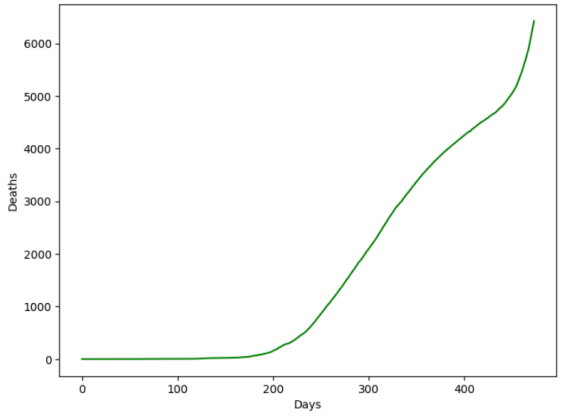

Fig3. Death graph of Kerala.

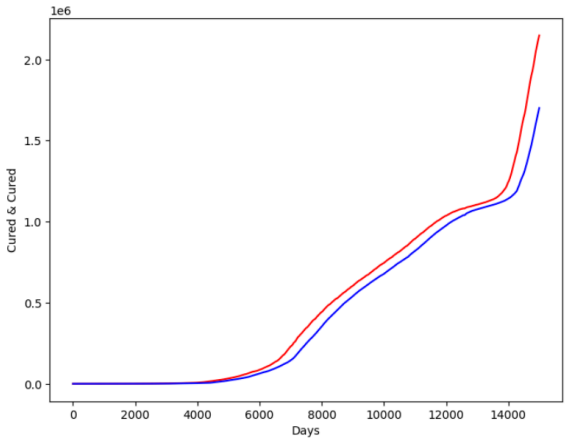

Fig4. Confirmed and Cured cases in Kerala

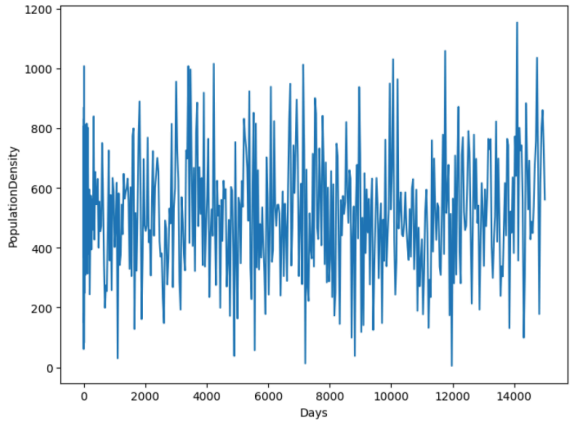

Fig5. Graph of populationDensity in Kerala

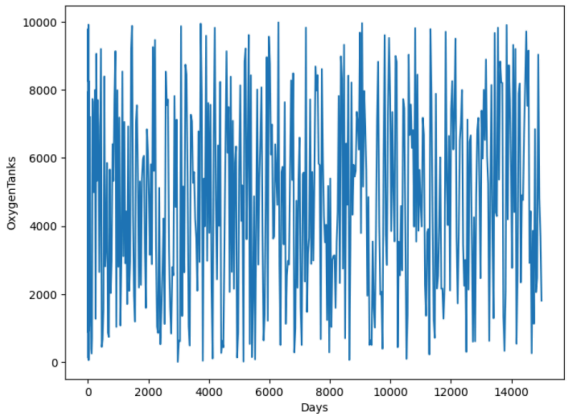

Fig6. Graph of OxygenTanks in Kerala

From the above graphs, it is clear that the “populationDensity” and “OxygenTanks” are stationary through time whereas cured and confirmed cases have a similar trend to deaths. So to choose the features to be used to predict we further calculated the correlation between the features and “Deaths”.

```
Cured 0.9984530476941744
Deaths 1.0
Confirmed 0.9896221176913746
OxygenTanks 0.0809828175805085
PopulationDensityPerSqKm 0.04243701956374865
```

It is clear that “Cured” and “Confirmed” have a high correlation, unlike other features. Therefore we decided to use only “Cured” and “Confirmed” for prediction.

Prediction model using Deep-Learning

We used many deep-learning models and finally found that LSTM works very well with the data. Initially, our idea was to group states that have similar patterns and ranges and use unique models for each group. This worked very well with the given data, but few outliers like "Mizoram", "Sikkim", "Nagaland", "Arunachal Pradesh", "Lakshadweep", "Dadra and Nagar Haveli and Daman and Diu", "Andaman and Nicobar Islands", "Tripura", "Manipur", "Meghalaya", "Ladakh" didn't work very well the models. This is where we got the idea to normalize the data individually for every state.

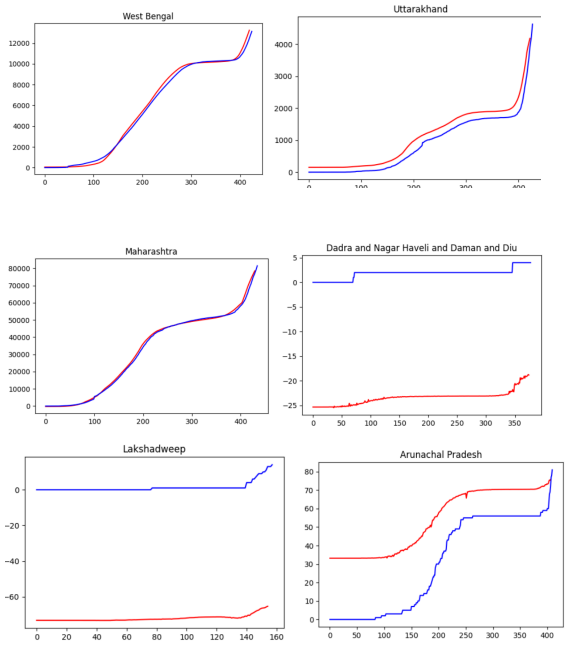

Fig 7. Predicted vs actual graph of various states

To overcome this problem we implemented our new idea and the results were better than the previous versions. We used a hashtable to store the state-wise data along with the respective scalers

```
state_data_dict = {}
```

```
for state in Total_data_prime["State/UnionTerritory"].unique():
    sc2 = MinMaxScaler(feature_range=(0,1))
    sc1 = MinMaxScaler(feature_range=(0,1))
    df = Total_data_prime[Total_data_prime["State/UnionTerritory"] == state]
    death_df = data[data["State/UnionTerritory"] == state]
    df[["Cured", "Confirmed"]] = sc2.fit_transform(df[["Cured", "Confirmed"]].values)
    death_df[["Deaths"]] = sc1.fit_transform(death_df[["Deaths"]].values)
    state_data_dict[state] = [df, death_df, sc2, sc1]
```

To train the data we used a single deep learning model for every state. Then predicted the test data using the model built and then inverse transformed the values using the stored scalers in the hashtable.

```
early_stopping = EarlyStopping(monitor='loss', patience=10)

model = Sequential([Bidirectional(layers.LSTM(44, return_sequences=True),input_shape=(x,2)),
                    layers.LSTM(8),
                    layers.Dense(16, activation = "relu"),
                    layers.Dense(1)])

model.compile(optimizer=Adam(learning_rate = 0.001),loss='mae', metrics = ['mean_absolute_error', 'accuracy'] )
model.fit(X_train1,y_train,epochs=200, batch_size=100, callbacks=[early_stopping], verbose=0)
```

Then we found that the initial 100 to 200 days of each state does not affect the future values so we trained the models using the past 200 to 300 days. By doing this our model performed better than the previous and reduced the MAPE score significantly.

```
for state in state_data_dict:
    if state in ["Maharashtra", "Delhi", "Uttar Pradesh", "Punjab", "Chandigarh", "Gujarat", "Madhya Pradesh"]:
        death_x = state_data_dict[state][1].iloc[300:,:]
        df_x = state_data_dict[state][0].iloc[300:,:]
        #print(df_x)
        scaler = state_data_dict[state][3]
        data_scaled = df_x.iloc[:len(death_x),:]
        m1, v = model_groupx(data_scaled, death_x)
        model_dict[state] = [m1, v]
        X, Y = to_test(df_x, death_x, v)
        to_predict(m1, X, Y, state, scaler)
    else:
        death_x = state_data_dict[state][1].iloc[150:,:]
        df_x = state_data_dict[state][0].iloc[150:,:]
        #print(df_x)
        scaler = state_data_dict[state][3]
        data_scaled = df_x.iloc[:len(death_x),:]
        m1, v = model_groupx(data_scaled, death_x)
        model_dict[state] = [m1, v]
        X, Y = to_test(df_x, death_x, v)
        to_predict(m1, X, Y, state, scaler)
```

We iterated this approach with new sets of layers in the model to make sure the model does not overfit and produce future values that do not drop over time since the data is cumulative it is impossible for the values to go down.

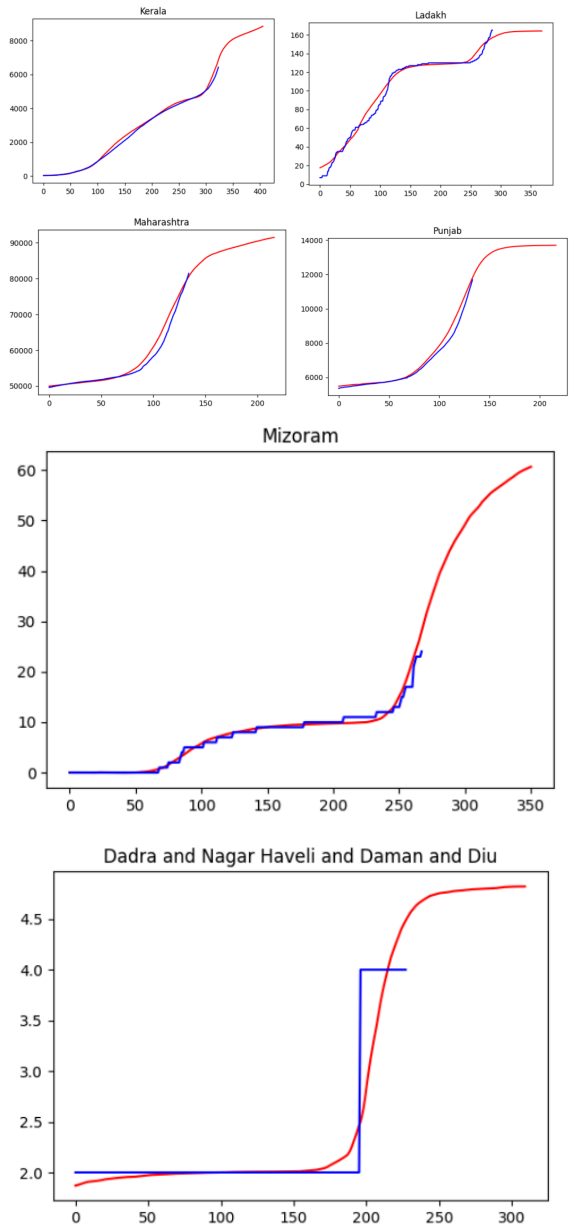

Fig8. Predicted vs actual of various states v2

Stored the predicted in a CSV file and saved and submitted in the Kaggle submissions.

Model Architecture

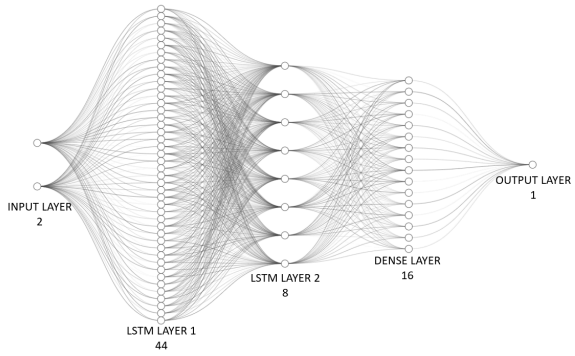

Fig9. Deep learning model architecture.

Conclusion

Our initial idea got a MAPE score of 6957.00 and we increased the accuracy by a few minor changes in the older version and got a MAPE score of 3134.00. And finally with our new approach, we reduced the error even further to 1724.00.

We understood that to make a better predictive model we should understand the nature of the data and should try to break down the problem into simpler subproblems. At first, we tried to fit the model with the entire dataset without splitting them by states, once we understood the data it was much more simple than the start. It is fascinating to realize that this dataset can be trained just by using a simple LSTM model with a very small number of neurons and lags.

Google Colab Notebooks

[1] Data PreProcessing and visualization of kerala
[2] Predictive model by grouping the states
[3] Normalizing the data by group
[4] Normalizing the data by state (Final approach)