Prediction the Global Spread of COVID-19

On 11 March 2020, the World Health Organization (WHO) officially classified COVID-19 as a pandemic. At the time, the number of confirmed cases were over 130,000 across 114 countries around the world, with the number of deaths caused by the disease over 5,000. The objective of this project is to build an epidemiological model that predicts the spread of COVID-19 throughout the world. The target variable is the cumulative number of deaths caused by COVID-19 in each country by each date.

1 Data

The train data contains data from 22 January to 5 March 2020. "Target" is the number of confirmed deaths and "Cases" is the number of reported infections.

[1]:
```
import pandas as pd
data=pd.read_csv('C:/Users/Rayen/Desktop/AI/train.csv')
predf=pd.read_csv('C:/Users/Rayen/Desktop/AI/SampleSubLocal.csv')
predt=pd.read_csv('C:/Users/Rayen/Desktop/AI/SampleSubmission1.csv')
```

[29]:
```
data['Territory']=data['Territory X Date'].str.split('X',1).str[0].str.strip()
data['Date']=pd.to_datetime(data['Date'])
dic=data['target'].to_dict()
```

[2]:
```
data.head()
```

[2]:
```
        Territory X Date  target  cases    Territory     Date
0  Afghanistan X 1/22/20       0      0  Afghanistan  1/22/20
1  Afghanistan X 1/23/20       0      0  Afghanistan  1/23/20
2  Afghanistan X 1/24/20       0      0  Afghanistan  1/24/20
3  Afghanistan X 1/25/20       0      0  Afghanistan  1/25/20
4  Afghanistan X 1/26/20       0      0  Afghanistan  1/26/20
```

[30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

```
import itertools
import math
from statsmodels.tsa.arima_model import ARIMA
import warnings
%matplotlib inline
```

2 Hyperparameters Tuning

[31]:
```
from sklearn.metrics import mean_squared_error
"""
Function that finds the best parameters for the Arima model. The metric used to␣
 ↪evaluate the models is the Mean Squared error.
"""
def find_best_arima(train):
    warnings.filterwarnings("ignore")
    train = train.astype('float32')
    best_score = float("inf")
    best_cfg=[]
    p_values = [0, 1, 2,3]
    d_values= [0,1,2,3]
    q_values= [0,1,2,3]
    predictions =[]
    train_size = int(len(train)* 0.66 )
    train_set, test_set = train[0:train_size], train[train_size:] #data splitting
    history = [x for x in train_set]

    param=(0,0,0) ##
    best_trend=[]
    best_method=[]
    for p in p_values:
        for d in d_values:
            for q in q_values:

                param = (p,d,q)
                try:
                    train_size = int(len(train)* 0.66 )
                    train_set, test_set = train[0:train_size], train[train_size:]
                    history = [x for x in train_set]
                    predictions =list()

                    for t in range(len(test_set)):
                        mod = ARIMA(history, order = param)
                        results = mod.fit(disp=False)
                        yhat = results.forecast()[0]
                        predictions.append(yhat)
                        history.append(test_set[t])
                    error = mean_squared_error(test_set, predictions)
                    if (error < best_score):
                        best_score, best_cfg = error,param
                        print('ARIMA%s MSE=%.3f ' % (best_cfg,error)) ##
                except:
                    continue

    print(best_cfg)
    print(best_score)

    best_model = {
                  'order':best_cfg,
                  'mse':best_score
                 }

    return best_model
```

[32]:
```
"""
Forecasting the number of deaths in each country for the next two months
"""
def prediction (order,series):

        series = series.astype('float32')
        model = ARIMA(series, order=order)
        model_fit = model.fit(disp=False)
        output = model_fit.forecast(steps=67) #steps=number of days to predict
        yhat = output[0]

        return(yhat)
```

3 Forecasting

[34]:
```
predictions=[]

"""
Forecasting the number of deaths due to COVID-19 in each country for the next␣
 ↪two months
"""
for i in data['Territory'].unique():
    start_date='2020-03-06'
    end_date='2020-04-01'
    print("**********************"+i+"**********************")
    x=data.loc[data['Territory']==i]
    x=x[['Date','target']]

    mask = (x['Date'] >= start_date) & (x['Date'] <= end_date)
    z = x.loc[mask]

    x.index=x['Date']
    x=x.drop(columns=['Date'])
    series = x
    best_model=find_best_arima(series.values)
    print('best config is: ARIMA%s MSE=%.3f  '%␣
↪(best_model['order'],best_model['mse']))
    yhat=prediction(best_model['order'],series.values)

    for r in z['target'].tolist():
        predictions.append(r)

    for k in yhat :
        predictions.append(k)
```

```
**********************Afghanistan**********************
ARIMA(0, 0, 0) MSE=4.209
ARIMA(0, 1, 0) MSE=0.235
(0, 1, 0)
0.23457294944010035
best config is: ARIMA(0, 1, 0) MSE=0.235
**********************Albania**********************
ARIMA(0, 0, 0) MSE=32.710
ARIMA(0, 1, 0) MSE=1.210
(0, 1, 0)
1.209949939035383
best config is: ARIMA(0, 1, 0) MSE=1.210
**********************Algeria**********************
ARIMA(0, 0, 0) MSE=385.326
ARIMA(0, 1, 0) MSE=13.153
ARIMA(0, 2, 0) MSE=3.948
(0, 2, 0)
3.9480842879707323
best config is: ARIMA(0, 2, 0) MSE=3.948
**********************Andorra**********************
ARIMA(0, 0, 0) MSE=17.508
ARIMA(0, 1, 0) MSE=1.429
ARIMA(0, 2, 0) MSE=1.135
(0, 2, 0)
1.1350158059488573
best config is: ARIMA(0, 2, 0) MSE=1.135
**********************Angola**********************
ARIMA(0, 0, 0) MSE=0.613
ARIMA(0, 1, 0) MSE=0.160
(0, 1, 0)
...
```

- Rounding the numbers to have a better accuarcy

[44]:
```
predictionsf= []
for i in predictions:
    k=i//1
    if (i<0):
        predictionsf.append(0)

    elif ( i-k >0.15):
        predictionsf.append(int(k+1))
    elif (i-k>0.4):
        predictionsf.append(int(k+2))
    elif (i-k>0.7):
        predictionsf.append(int(k+3))
    elif (i-k>0.9):
        predictionsf.append(int(k+4))
    else:
        predictionsf.append(int(k))
print(predictionsf)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 4, 4, 4, 4, 4, 4,
4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2,
2, 2, 2, 2, 4, 5, 5, 6, 8, 10, 10, 11, 15, 15, 16, 16,...]
```

[52]:
```
for i in range(len(predt)):
    predt.loc[i,'target']=predictionsf[i]
```

- The final prediction:

[53]:
```
predf.head()
```

[53]:
```
        Territory X Date  target
0  Afghanistan X 4/2/20       0
1  Afghanistan X 4/3/20       0
2  Afghanistan X 4/4/20       0
3  Afghanistan X 4/5/20       0
4  Afghanistan X 4/6/20       0
```

[55]:
```
predt=predt[['Territory X Date','target']]
```

[56]:
```
predt.to_csv("submission11.csv",index=False) #Converting the results from a␣
 ↪dataframe to CSV
```