BIOLOGICALLY INSPIRED ARTIFICIAL INTELLIGENCE



Predykcja rozwoju choroby

Krzysztof Kuczyński

Jakub Sądel

INF GKiO3

Wstęp

Celem naszego projektu było stworzenie programu który będzie mógł przewidzieć rozwój chorób zakaźnych na świecie. W naszym przypadku zaczniemy od popularnego ostatnio Covid-19.

Przygotowanie danych

Nasz zbiór z danym został znaleziony na [kaggle.com](https://www.kaggle.com/optimo/covid19-enriched-dataset-week-2#Load-Covid-Data) i składa się z 18816 rzędów(294 krajów i regionów przez kolejne 64 dni) i 38 różnych zmiennych takich jak populacja, ilość zarażonych lub odsetek palaczy.

|  |
| --- |
| #Importing data from github  import pandas as pd  path='https://raw.githubusercontent.com/sondisonda/covid19SpreadForecast/master/enriched\_covid\_19\_week\_2.csv'  df = pd.read\_csv(path) |

Po załadowaniu danych zauważono że niektóre zmienne można usunąć np. ‘id’ bo się dubluje z domyślnym, ‘Date’ ponieważ dane są poindeksowane w odpowiedni sposób, lub ‘Province\_State'

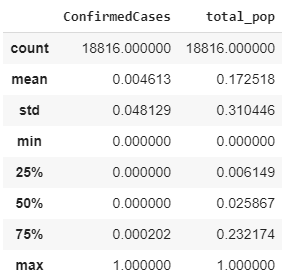
|  |
| --- |
| #Removing unnecessary columns  df.drop(['Id'], 1, inplace=True)  df.drop(['Province\_State'], 1, inplace=True)  df.drop(['Date'], 1, inplace=True) |

Po sprawdzeniu czy w tabeli nie pojawiły się jakieś nieprawidłowości nadszedł czas na podzielenie zbioru na treningowy i testowy. Trzeba pamiętać o odpowiednim podzieleniu zbioru, tak aby nie podzielić danych krajowych jednego państwa w zbiór treningowy i szkoleniowy.

|  |
| --- |
| #Spliting data in a training set and test set  lines\_count = df.shape[0]  lines\_per\_country = 64  country\_count = int(lines\_count / lines\_per\_country)  train\_country\_count = int(country\_count \* (2/3))  test\_country\_count = int(country\_count \* (1/3))  train\_rows\_count = int(lines\_count \* (2/3))  test\_rows\_count = int(lines\_count \* (1/3))  print('Country count: ' + str(country\_count))  print('Train rows count: ' + str(train\_rows\_count))  print('Test rows count: ' + str(test\_rows\_count))  print('Train country count: ' + str(train\_country\_count))  print('Test country count: ' + str(test\_country\_count)) |
| Country count: 294  Train rows count: 12544  Test rows count: 6272  Train country count: 196  Test country count: 98 |

By uprościć odbiór wyniku wykonujemy jeszcze operacje uproszczenia danych gdzie zostawiamy 2 najważniejsze kolumny, a następnie wykonujemy normalizacje w zakresie od 0 do 1.

|  |
| --- |
| #Data simplification  simplified\_df = df[['ConfirmedCases','total\_pop']]  simplified\_df.shape |
| #Data normalization  from sklearn.preprocessing import MinMaxScaler  from numpy import array  scaler = MinMaxScaler()  reshaped = array(simplified\_df.ConfirmedCases.values).reshape(-1, 1)  scaler.fit(reshaped)  normalized = scaler.transform(reshaped)  simplified\_df['ConfirmedCases'] = normalized  reshaped = array(simplified\_df.total\_pop.values).reshape(-1, 1)  scaler.fit(reshaped)  normalized = scaler.transform(reshaped)  simplified\_df['total\_pop'] = normalized  simplified\_df.describe() |



Wygląd danych po tych operacjach

Dane w takiej postaci zapisujemy do odpowiednich zmiennych.

|  |
| --- |
| #Data assignment  x\_train = simplified\_df.head(train\_rows\_count)  x\_test = simplified\_df.tail(test\_rows\_count)  y\_train = x\_train['ConfirmedCases']  y\_test = x\_test['ConfirmedCases'] |

Dla każdego kraju istnieje ciąg 64 kolejnych dni, który dzielimy na podciągi z 63 krokami. Ostatni krok czasowy będzie używany do prognozowania szkolenia tj : będziemy trenować model do przewidywania liczby przypadków w 64 dniu zgodnie z poprzednimi 63 dniami.

LSTM potrzebuje danych w formacie [próbki, przedziały czasowe i cechy]. Tutaj mamy 196 próbek, 63 kroki czasowe na próbkę i 2 funkcje. Przed stworzeniem modelu musimy przygotować dla niego odpowiednie dane. Dla testów robimy to analogicznie jak do treningu.

|  |
| --- |
| #Preparing train data  import numpy as np  from numpy import array  x\_days\_train = list()  y\_days\_train = list()  length = lines\_per\_country  country\_count = train\_country\_count  for i in range(0,length\*country\_count,length):      x\_day\_train = x\_train[i:i+length-1]      x\_days\_train.append(x\_day\_train.values)      y\_day\_train = y\_train[i+length-1:i+length]      y\_days\_train.append(y\_day\_train.values)  x\_train = array(x\_days\_train)  print(x\_train.shape)  y\_train = array(y\_days\_train)  print(y\_train.shape)  x\_train = x\_train.reshape(train\_country\_count, 63, 2)  print(x\_train.shape) |

W naszym projekcie wykorzystaliśmy bibliotekę keras czyli wysoko poziomowego API działającego na tensorflow. Nasz model jest zwykłym modelem sekwencyjnym co oznacza warstwy następują po sobie, w tym wypadku są to rekurencyjna warsta Long Short-Term Memory i „domyślna” Dense. Po dodaniu odpowiednich warstw kompilujemy go, a następnie trenujemy i oceniamy.

|  |
| --- |
| #Creating and training  model  from keras.models import Sequential  from keras.layers import LSTM  from keras.layers import Dense  n\_steps = 63  n\_features = 2  n\_batch = 4  model = Sequential()  model.add(LSTM(16, input\_shape=(n\_steps, n\_features)))  model.add(Dense(1))  model.compile(optimizer='adam', loss='mse')  history = model.fit(x\_train, y\_train, epochs=50, batch\_size=n\_batch, validation\_data=(x\_test, y\_test),verbose=0, shuffle=False) |
| #Evaluating model  import matplotlib.pyplot as plt  plt.plot(history.history['loss'], label='train loss')  plt.plot(history.history['val\_loss'], label='test loss')  plt.legend()  plt.show() |
|  |

Na samym końcu porównujemy otrzymane wyniki i zapisujemy model.

|  |
| --- |
| #Comparing results  import matplotlib.pyplot as plt  predictions = model.predict(x\_test)  plt.figure(figsize = (10,5))  plt.plot(range(len(predictions)),predictions, label="Predicted Cases")  plt.plot(range(len(y\_test)),y\_test, label="Real Cases")  plt.ylabel('Deads',fontsize=15)  plt.legend(loc="upper left")  plt.show() |
|  |
| #Model save  model.save("model.h5") |