Report on Unsupervised Learning with COVID-19 dataset

Akrit Woranithiphong 635810456

April 29th 2022

1 Introduction

1.1 Dataset

The dataset is a big COVID-19 dataset consisting of 49.5MB with 176915 rows and 67 columns, which consist of 242 countries encoded in the ["iso-code"]. The dataset can be derived from https://github.com/owid/covid-19-data/tree/master/public/data.

1.2 Objective

The main objective of this research is to find a decent cluster using unsupervised machine learning techniques. Clustering methods include DBSCAN, Mean Shift Clustering, Hierarchical Clustering, and Self Organizing Maps.

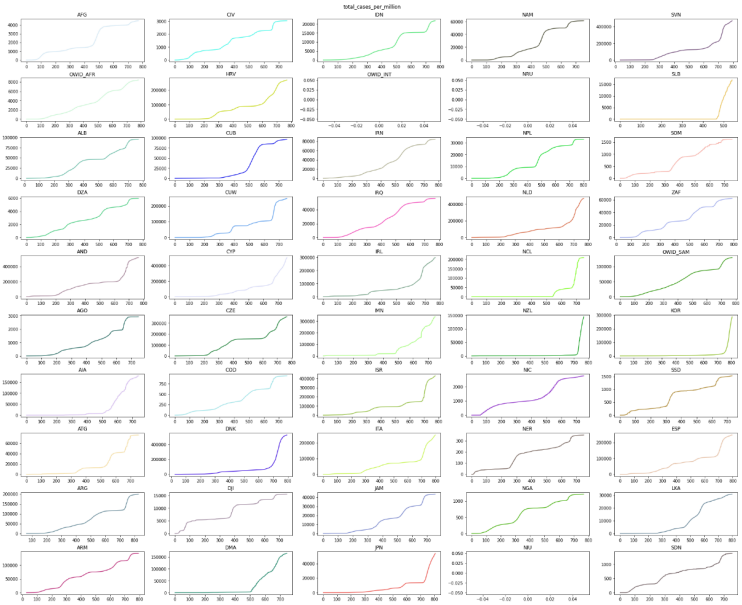
2 Data Processing

2.0.1 Separate each country from the DataFrame

The dataset is encoded in a DataFrame which is hard to work with. Since we will be working with high dimensions of data good format to easily work with is in NumPy array. First, we will transform a DataFrame into a List of DataFrame, separating them by country or as known as the "iso-code."

2.0.2 Visualizing Data

One of the best ways to understand time series data is to visualize them in a plot. In the example below, we will be looping through the dataset and plotting each column in the dataset to understand our data. Although the image below only shows one column, which is less than a fraction of what is done, the full plot is included in the full code under the appendix.



2.0.3 Data Preparation

After we've got understood dataset, we prepared the data in an accessible format to apply clustering techniques. First, Features Extraction, we picked only column which scaled per million and data that 80% of all countries have filled. Second, Applied Padding, we've done this by calculating the mean length of time series and applied padding of 0 for those lower than average and slicing to the mean for those above average. Third, Fill NA, from understanding the data, we know that if the data is 0 then the row is placed with NA. To perform cluster we replaced NA back to 0 by replacing it. Lastly, we normalized the data by dividing by a million, now our data is between 0 and 1.

2.0.4 Dimensionality Reduction

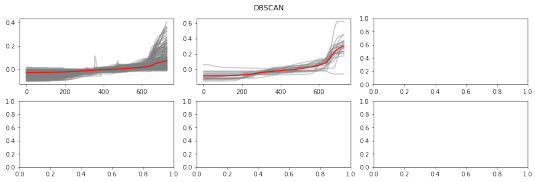
Dimensionality Reduction is used to reduce the dimension/size of the data. In the given dataset the shape is (242, 731, 5). We perform PCA to reduce the dimension to (242, 731). PCA is a great choice for this since it retain the variance of the data and keep most of the information intact. In this dataset performing reduction from 5 arrays to 1 array retain 0.9609% of the variance, which is decent.

2.0.5 Clustering

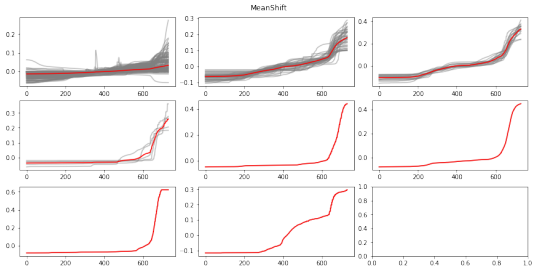
Clustering method is DBSCAN, Mean Shift Clustering, Hierarchical Clustering and Self Organizing Maps because these clusters don't need to specified k number of cluster.

3 Clusters

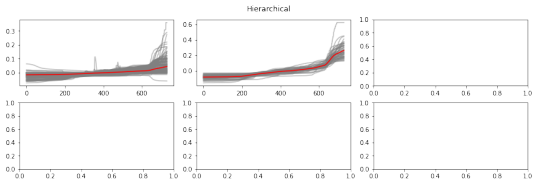
3.0.1 DBSCAN



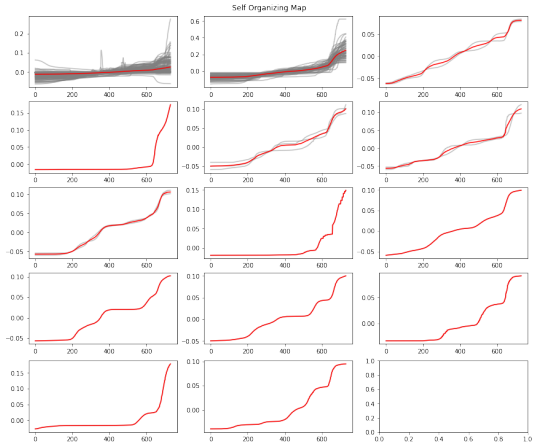
3.0.2 Mean Shift Clustering



3.0.3 Hierarchical Clustering



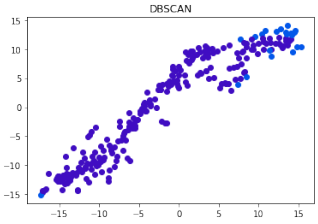
3.0.4 Self Organizing Maps



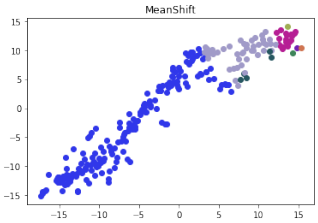
4 Visualizing Cluster in 2D

To visualize the clusters in 2D, we reshaped the data from (242, 731) to (242) by performing a Dimensionality Reduction technique; t-SNE. t-SNE is a good choice because it separates points from each other, therefore making our more distinguishable.

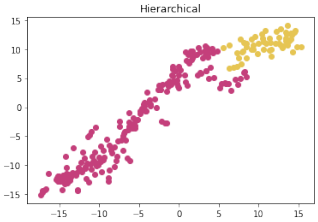
4.0.1 DBSCAN



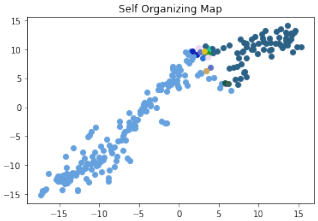
4.0.2 Mean Shift Clustering



4.0.3 Hierarchical Clustering



4.0.4 Self Organizing Maps



5 Acknowledgement & Codes

5.0.1 Acknowledgement

All codes are written in JupyterNotebook format, and each column is separated by "%%." Python version is 3.9.7 which supported type annotation. Please install all the given dependencies to avoid errors.

5.0.2 Codes

```
# %% [markdown]
# # Dependencies & Import

# %%
# # Python version 3.9.7
# !pip install pandas
# !pip install numpy
# !pip install tensorflow
# !pip install keras
# !pip install matplotlib
# !pip install sklearn
# !pip install minisom
# !pip install scipy

# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
from typing import *
import multiprocessing
import random
import math

import warnings
warnings.filterwarnings("ignore")

# %% [markdown]
# # View the dataset

# %%
df: pd.DataFrame = pd.read_csv("owid-covid-data.csv")

df
```

```
# %% [markdown]
# View all na

# %%
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
# more options can be specified also
    print(df.isna().sum())

# %% [markdown]
# Create a list of all countries

# %%
# preserve order and filter out duplicate
listOfCountry: List[str] = list(dict.fromkeys(df['iso_code']))

print(listOfCountry)
print("listOfCountry len is: %i \n" % len(listOfCountry))

# %% [markdown]
# Split Dataframe into lists of dataframe per country

# %%
listOfDf: List[pd.DataFrame] = []
for country in listOfCountry:
    listOfDf.append(df[df['iso_code'] == country])

listOfDf

# %% [markdown]
# # Visualizing the data

# %%
def plot_all(list_df: List[pd.DataFrame],
             fig_size_mul: float=2,
             fig_aspect_ratio: float=2.5,
             row_len: int=5,
             col: str='total_cases_per_million') -> None:

    col_len: int = math.ceil(len(list_df) / row_len)

    fig, ax = plt.subplots(col_len,
                           row_len,
                           figsize=(row_len * fig_size_mul * fig_aspect_ratio,
                           constrained_layout=True)

    for i in range(col_len * row_len):
```

```
            _x = i % col_len
            _y = i // col_len

            if i >= len(list_df):
                break

            _df = list_df[i]

            if len(_df) > 1:
                ax[_x][_y].plot(range(len(_df[col])), _df[col], c=np.random.rand(3,)
                ax[_x][_y].set_title(list(_df['iso_code'])[0])

    fig.suptitle(col)
    plt.show()

# %%
plot_all(listOfDf, col='total_cases_per_million')
plot_all(listOfDf, col='total_deaths_per_million')
plot_all(listOfDf, col='total_deaths_per_million')
plot_all(listOfDf, col='new_vaccinations_smoothed_per_million')
plot_all(listOfDf, col='icu_patients_per_million')
plot_all(listOfDf, col='hosp_patients_per_million')
plot_all(listOfDf, col='weekly_icu_admissions_per_million')
plot_all(listOfDf, col='weekly_hosp_admissions_per_million')
plot_all(listOfDf, col='new_vaccinations_smoothed_per_million')
plot_all(listOfDf, col='excess_mortality_cumulative_per_million')

# %% [markdown]
# # Data preparation

# %%
def format_list(list_df: List[pd.DataFrame],
                selected_column: List[str]=None,
                fill_na: bool= False,
                use_padding: bool=False,
                to_list:bool=False,
                normalize: bool=False) -> np.array:

    if use_padding:
        list_of_len = [] # count len on each df
        for df in list_df:
            list_of_len.append(len(df))
        mean_len: int = int(sum(list_of_len) / len(list_of_len)) # get average o

    new_list = []
    for _, df in enumerate(list_df):
```

```
        if selected_column:
            df: pd.DataFrame = df[selected_column]
        if fill_na:
            df: pd.DataFrame = df.fillna(0)
        if to_list:
            df: List[int, float, str] = df.values.tolist()
        if use_padding:
            if len(df) >= mean_len: # better than average
                df: List[int, float, str] = df[-mean_len:]
            else:
                while len(df) < mean_len: # worst than average
                    padding_list: List[int] = [0 for i in range(len(selected_colu
                    df: List[int, float, str] = [padding_list] + df # recursivel

        if normalize:
            processed_df: List[List[int, float]] = []
            for i in df:
                i_row: List[int, float] = []
                for x in i:
                    i_row.append(x / 1000000) # normalize by mil
                processed_df.append(i_row)
        else:
            processed_df: List[List[int, float, str]] = df

        new_list.append(processed_df)


    return np.array(new_list)


formatted_data: np.array = format_list(list_df=listOfDf,
                           selected_column=['total_cases_per_million',
                                            'new_cases_per_million',
                                            'total_deaths_per_million',
                                            'new_deaths_per_million',
                                            'new_vaccinations_smoothed_per_millio
                           fill_na=True,
                           use_padding=True,
                           to_list=True,
                           normalize=True)

print('formatted_data.shape : %s\n' % str(formatted_data.shape))

# %% [markdown]
# # Dimensionality Reduction for Clustering
# PCA (242, 731, 5) -> (242, 731)
```

```
# %%
from sklearn.decomposition import PCA

pca_formatted_data: List[List[List]] = []
pca = PCA(n_components=1)
for eachData in formatted_data:
    newData: np.array([int, float]) = pca.fit_transform(eachData)
    pca_formatted_data.append(newData.reshape(-1, 1))

pca_formatted_data: np.array = np.array(pca_formatted_data)
print('PCA Variance Retain :  %.4f%%\n' % pca.explained_variance_ratio_[0])

reshaped_formatted_data: np.array = pca_formatted_data.reshape(pca_formatted_dat

# %% [markdown]
# # Building Model (Clustering)
# Dimension (242, 731)

# %% [markdown]
# DBSCAN

# %%
from sklearn.cluster import DBSCAN

dbscan = DBSCAN()
dbscan_clusters = dbscan.fit(reshaped_formatted_data)
dbscan_number_of_cluster = len(set(dbscan_clusters.labels_))
dbscan_cluster_labels = abs(dbscan_clusters.labels_)
print('DBSCAN Number of Clusters : %s' % len(set(dbscan_clusters.labels_)))
print('DBSCAN Bin : %s\n' % np.bincount(dbscan_cluster_labels))

# %% [markdown]
# MeanShift

# %%
from sklearn.cluster import MeanShift

meanshift = MeanShift()
meanshift_clusters = meanshift.fit(reshaped_formatted_data)
meanshift_number_of_cluster = len(set(meanshift_clusters.labels_))
meanshift_cluster_labels = abs(meanshift_clusters.labels_)
print('MeanShift Number of Clusters : %s' % len(set(meanshift_clusters.labels_)))
print('MeanShift Bin : %s\n' % np.bincount(meanshift_cluster_labels))

# %% [markdown]
```

```
# Hierarchical Clustering

# %%
from sklearn.cluster import AgglomerativeClustering

hierarchical = AgglomerativeClustering()
hierarchical_clusters = hierarchical.fit(reshaped_formatted_data)
hierarchical_number_of_cluster = len(set(hierarchical_clusters.labels_))
hierarchical_cluster_labels = abs(hierarchical_clusters.labels_)
print('Hierarchical Number of Clusters : %s' % len(set(hierarchical_clusters.lab
print('Hierarchical Bin : %s\n' % np.bincount(hierarchical_cluster_labels))

# %% [markdown]
# Self Organizing Maps (SOM)

# %%
from minisom import MiniSom

som_x: int = 1 + int(math.sqrt(5*math.sqrt(len(reshaped_formatted_data))))
som_y: int = 1 + int(math.sqrt(5*math.sqrt(len(reshaped_formatted_data))))
som = MiniSom(som_x, som_y, len(reshaped_formatted_data[0]), random_seed=42, sigm
som.random_weights_init(reshaped_formatted_data)
som.train(reshaped_formatted_data, 10000)
som_win_map: Dict[Any, Any] = som.win_map(reshaped_formatted_data)
som_bin:List[int] = []
for i in list(som_win_map):
    som_bin.append(np.array(som_win_map[i]).shape[0])
som_clusters = [som.winner(reshaped_formatted_data[i]) for i in range(len(reshap
som_cluster_labels = [list(som_win_map).index(i) for i in som_clusters]
print('Self Organizing Maps Number of Clusters : %s' % len(set(som_cluster_label
print('Self Organizing Maps Bin : %s\n' % som_bin)

# %% [markdown]
# # Visualizing Cluster

# %%
def plot_cluster(reshaped_formatted_data: np.array, clusters_labels: List[int],
    cluster_num = len(set(clusters_labels))
    plt_col: int  = 3
    plt_row: int  = ((cluster_num - 1) // 3) + 1
    if plt_row < 2:
        plt_row: int  = 2
    fig, axs = plt.subplots(plt_row, plt_col, figsize=(plt_col * 4, plt_row * 2)
    for i in range(cluster_num):
        row: int = i // 3
        col: int = i % 3
```

```
        curr_cluster: List[List] = []
        for _i, x in enumerate(clusters_labels):
            if i == x:
                axs[row][col].plot(reshaped_formatted_data[_i], c='gray', alpha=
                curr_cluster.append(reshaped_formatted_data[_i])
        axs[row][col].plot(np.average(curr_cluster, axis=0), c='r')

    fig.suptitle(cluster_method_name)
    plt.show()

# %%
plot_cluster(reshaped_formatted_data, dbscan_cluster_labels, 'DBSCAN')
plot_cluster(reshaped_formatted_data, meanshift_cluster_labels, 'MeanShift')
plot_cluster(reshaped_formatted_data, hierarchical_cluster_labels, 'Hierarchical
plot_cluster(reshaped_formatted_data, som_cluster_labels, 'Self Organizing Map')

# %% [markdown]
# # Visualizing Cluster in 2D

# %%
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
visualizable_data = tsne.fit_transform(reshaped_formatted_data)

def plot_2d(visualizable_data: np.array, clusters: List[int], cluster_method_lab
    all_colors: List[str] = [np.random.rand(3,) for i in range(100)]
    random.shuffle(all_colors)
    plot_colors: List[str] = list(all_colors)[:len(set(clusters))]

    for i, cluster in enumerate(clusters):
        plt.scatter(visualizable_data[i][0], visualizable_data[i][1], color=plot

    plt.title(cluster_method_label)
    plt.show()

# %%
plot_2d(visualizable_data, dbscan_cluster_labels, 'DBSCAN')
plot_2d(visualizable_data, meanshift_cluster_labels, 'MeanShift')
plot_2d(visualizable_data, hierarchical_cluster_labels, 'Hierarchical')
plot_2d(visualizable_data, som_cluster_labels, 'Self Organizing Map')
```