Proyecto: COVID-19

31508811-1 Martiñón Luna Jonathan José.
31506842-1 Ortega Ibarra Jaime Jesus.
41800471-9 Tapia López José de Jesús.

I. Objetivos

- **Rayos X:**
  Dado dos conjunto de imágenes de rayos X del tórax de algunas personas; en las que el primero es de pacientes que se contagiaron de coronavirus, mientras que el segundo es de personas normales (sanas), utilizar modelos de clasificación supervisada para predecir si una persona padece la enfermedad del COVID-19 o no.
- **Tomografías Axiales pulmonares**:
  Se poseen imágenes de tomografías axiales de pacientes que tienen COVID-19. Se pretende comparar los daños presentes en los pulmones, comparándo con tomografías de pulmones que no tengan COVID-19. Esto no quiere decir que sean pulmones sanos, simplemente no tienen COVID-19
- **Identificar Regiones:**
  A partir de las tomografías conocidas,que tienen COVID-19, buscaremos identificar las diversas regiones: Esmeralizado, consolidado y derrame pleural.

II. Ambiente

La práctica fue desarrollada en *Google Colab*, haciendo uso de Python 3.

III. Introducción

COVID-19, también conocida como enfermedad por coronavirus o incorrectamente como neumonía por coronavirus. Es una enfermedad infecciosa causada por el virus SARS-CoV-2 y se detectó opor primera vez en la ciudad China de Wuhan (provincia de Hubei) en diciembre de 2019. Este padecimiento ha llegado a más de 100 territorios, causando que el 11 de marzo de 2020 la Organización Mundial de la Salud la declaró pandemia. En respuesta a la pandemia muchas organizaciones, universidades, centros de investigación e incluso gobiernos han creado bases de datos abiertas que contienen datos de personas que hayan contraído la enfermedad. Particularmente, para fines de este proyecto, nos van a interesar aquellas bases de datos que contengan la información en imágenes del tórax.

Este proyecto usará distintos algoritmos de minería de datos, los cuales explicamos brevemente.

A. Algoritmos

1) K-Means: También conocido como K-Medias, este algoritmo el analista establece el número de grupos; el algoritmo comienza asignando arbitrariamente los vectores del conjunto de datos $D = \{x_1, ..., x_n\}$ a k grupos; estos grupos iniciales se denotan como $\{A_1, ..., A_k\}$. Después, se calculan los centroides de cada grupo; estos centroides son medias multivariadas de las observaciones de cada grupo y cada centroide representa al grupo para el cálculo de las distancias. También es posible asignar arbitrariamente k medias iniciales y con ellas generar los grupos. Es muy poco probable que la configuración inical sea una buena solución, por tanto el algoritmo itera entre dos pasos:

1) Asignar cada observación al grupo más cercano
2) Recalcular los centroides de cada grupo

Si al menos una observación es reasignada en otro grupo, los centroides cambiarán y debe realizarse otra iteración; el algoritmo continuará pasando entre estos dos pasos hasta que ya no hay reasignaciones. En ese momento, cada observación pertenece al grupo que le es más cercano.

A partir de la configuración aleatoria inicial, la suma de cuadrados dentro de cada grupo ha sido minimizado; esto se debe a que dicha suma de cuadrados es equivalente a la suma de distancias euclideanas entre las observaciones y los centroides; además, mover cualquiera de las obervaciones incrementará la suma de distancias (y la suma de cuadrados dentro de los grupos). Por tanto, el algoritmo ha alcanzado una mejor asignación posible para las observaciones en los grupos y un mejor cálculo de los centroides.

Este algoritmo tiene un atractivo considerable porque minimza una función objetivo popular: la suma de cuadrados. Su inconveniente es que debe generar una configuración inicial aleatoria: una configuración diferente por lo general regresa un resultado distinto.

El centroide del grupo A_i es la media multivariada:

$\bar{x_i} = \frac{1}{n} \sum_{x_j \in A_i} x_j$, donde, n_i es el número de observaciones que pertenecen a A_i y $x_j = [x_{j,1}, ..., x_{j,h}]^T$; el número de atributos es h y el l-ésimo elemento de x_i es:

$\bar{x}_{i,l} = \frac{1}{n} \sum_{x_j \in A_i} x_{j,l}$, para $l = 1, ..., h$; la distancia entre cada observación x_j y un grupo A_i, se define como la distancia entre la observación y el centroide del grupo. Como $xi = [x_{i,1}, ..., x_{i,h}]^T$, la distancia euclideana cuadrada es:

$d_E^2(x_j, x_i) = \sum_{l=1}^{h} (x_{j,l} - x_{i,l})^2$.

Se puede utilizar la distancia cuadrada en lugar de la distancia, dado que el ordenamiento de más cercano a más distante será el mismo.

2) K Nearest Neighbor (KNN): Una observación con valor objetivo y_0 desconocido, se denota como $z_0 = (y_0, x_0)$, en donde el vector predictor x_0 de longitud p ha sido observado y se utilizará para predecir el valor de y_0.

La función de predicción $f(\cdot|D)$ se construye a partir del conjunto de datos $D = \{z_1, ..., z_n\}$; la notación condicional de $f(\cdot|D)$ enfatiza el papel del conjunto de entrenamiento D. La predicción de y_0 se denota como $\hat{y}_0 = f(x_0|D)$. Por tanto, se asume que la variable objetivo es una etiqueta que identifica la pertenencia a algún grupo; si el número de grupos es g, entonces por convención el conjunto de etiquetas es $0, 1, ..., g-1$.

Una función de predicción KNN, también conocida como K-vecinos más cercanos, utiliza un conjunto de observaciones en pares $D = \{(y_1, x_1), ..., (y_n, x_n)\}$ para los cuales se conocen todos los y_i valores objetivo; una observación objetivo es un par (y_0, x_0) para el que se desea calcular el valor de su variable objetivo y_0 . La versión más sencilla de esta función de predicción para problemas de clasificación opera determinando las k observaciones más cercanas (similares) a (y_0, x_0), basada en las distancias entre x_0 y x_i; el valor de la predicción será la etiqueta más entre las observaciones más cercanas: sus k-vecinos.

De manera más general, la predicción de y_0 con el algoritmo de k-vecinos más cercanos se obtiene determinando un vecindario de las k observaciones de entrenamiento más cercanas a x_0; después se calcula la proporción de los k vecinos que pertenecen al grupo j y se predice el valor de y_0 como el grupo con la mayor proporción entre los vecinos. Esta regla de predicción es equivalente a predecir que z_0 pertenece al grupo más común entre los k vecinos más próximos. Formalmente, la función de predicción que estima la probabilidad de pertenencia al grupo j se define como la proporción de los miembros del grupo j entre los k vecinos más cercanos: $P(y_0 = j|x_0) = \frac{n_j}{k}, j = 1, ..., g$, donde n_j es el número de vecinos dentro de los k más próximos que pertenecen a j.

De esta formna, las probabilidades de pertenencia estimadas se agrupan como un vector: $\hat{p}_0 = [P(y_0 = 1|x_0), ..., P(y_0 = g|x_0)]^T$.

El último paso para calcular la predicción obtiene el valor más grande dentro de $\hat{p}_0$: $f(x_0|D) = argmax(\hat{p}_0)$.

La función $argmax$ devuelve el índice del elemento más grande de un vector, en este caso, de $\hat{p}_0$; en el caso de existir empate entre varios valores, devolverá el primero de ellos. Sin embargo, es mejor idea romper los empates de otra forma, por ejemplo incrementando el valor de k en uno y recalculando $\hat{p}_0$.

3) Naive Bayes: La función de predicción bayesiana (inocente, naive) es un algoritmo conceptual y computacionalmente simple; se dice que es inocente debido a que se presupone que las características son independientes una de la otra.

El Clasificador Bayesiano es un clasificador probabilístico fundamentado en el teorema de Bayes y algunas hipótesis simplificadoras adicionales, generalmente es utilizado en teorias de la probabilidad y minería de datos. Dicho clasificador asume que la presencia o ausencia de una característica particular no está relacionada con la presencia o ausencia de cualquier otra característica, dada la clase variable.

Matemáticamente, si C_i representa la clase i, y $X = (X_1, ..., X_n)$ es el vector de características extraído para el píxel cuya clase tiene que ser encontrada, entonces la probabilidad de que el píxel pertenezca a una clase particular está dada por la probabilidad posterior $P(C_i|X)$. Entonces, la fórmula se basa en el teorema de Bayes:

$$P(Ci|X) = \frac{P(X|Ci)P(Ci)}{P(X)}, \tag{1}$$

donde $P(C_i)$ es la probabilidad a priori de la clase C_i; $P(X|C_i)$ es la probabilidad del vector de características X, dado que el píxel pertenece a la clase Ci; y, $P(X)$ es la probabilidad total del vector de características X (es decir, $\sum_i P(X|Ci)P(Ci)$). Un clasificador bayesiano primero calcula $P(Ci|X)$ usando la ecuación anterior. Luego, el clasificador da la etiqueta C_m a un vector de características X_0 dado si $P(Cm|X_0)$ es máximo, es decir, $C_m = argmax_i\{P(Ci|X)\}$. Las probabilidades anteriores $P(Ci), P(X)$ y la probabilidad condicional $P(X|Ci)$ se calculan a partir de las imágenes etiquetadas; sin embargo, dado que el término $P(X)$ aparece para cualquier clase C_i, es constante para cualquier i, por lo que se omite.

Para calcular las probabilidades anteriores, se debe asumir una distribución. Para las características discretas, las distribuciones multinomiales y de Bernoulli son populares.

Cuando se trata de datos continuos, una suposición típica es que los valores continuos asociados con cada clase se distribuyen de acuerdo con una distribución normal (o gaussiana).

De esta manera, tenemos que con base a la distribución, la fórmula para calcular $P(X|Ci)$ es la siguiente:

- *Bernoulli*: $\prod_k p_{i_k}^{x_k}(1 - p_{i_k})^{(1-x_k)}$.
- *Multinomial*: $\frac{(\sum_k x_k)!}{\prod_k x_k!}\prod_k p_{i_k}^{x_k}$, donde las muestras (vectores de características) representan las frecuencias con las que ciertos eventos han sido generados por un multinomio $(p_1, ..., p_n)$, y p_i es la probabilidad de que ocurra el evento i.
- *Gaussiano*: $\frac{1}{\sqrt{2\pi\sigma_i^2}}e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$.

4) Support Vector Machines (SVM): En español este algoritmo se conoce como Máquinas de soporte vectorial. Sirve para tratar con límites no lineales. Además, trabaja con *kernels*, los cuales son enfoques más eficientes utilizados para calcular la similitud entre 2 observaciones en una nuevo espacio dimensional. Tenemos 3 tipos de kernel's:

1) Lineal
   Usa el método de correlación de Pearson

$$K(x_i, x_i') = \sum_{j=1}^{P} x_{i,j}, x_{i,j}'$$

2) Polinómico
   Con grado *d* (*d*>1), permite una decisión más 'flexible'

$$K(x_i, x_i') = \left(1 + \sum_{j=1}^{P} x_{i,j}, x_{i,j}'\right)^d$$

3) Radial
   Sirve para datos locales, digamos, sólo serviría para aquellas observaciones parecidas a nuestro conjunto de entrenamiento.

$$K(x_i, x_i') = exp\left(-\gamma \sum_{j=1}^{P} (x_{i,j}, x_{i,j}')^2\right)$$

   Donde γ es una constante positiva que representa la flexibilidad del SMV, entre más grande más flexible.

B. Métricas

Después del resultado de una clasificación de registros, es posible asignarla a una de las siguientes 4 categorías:

- **Verdaderos positivos (VP)**: Son aquellos registros que han sido clasificados como *correspondientes* y que realmente sí lo son, dado que dichos registros se refieren a la misma entidad.
- **Falsos positivos (FP)**: Son aquellos registros que han sido clasificados como *correspondientes* pero NO lo son. El clasificador ha hecho una mala decisión.
- **Verdaderos negativos (VN)**: Son aquellos registros que han sido clasificados como *no correspondientes*, y que en efecto no lo son, dado que dichos registros hacen referencia a entidades diferentes.
- **Falsos negativos (FN)**: Son aquellos registros que han sido clasificados como *no correspondientes* y que de hecho sí son *correspondientes*, dado que los registros hacen referencia a la misma entidad, por lo tanto el clasificador ha hecho una mala decisión.

Una salida ideal de un proceso de clasificación es obtener la mayor cantidad de registros correspondientes como VP mientras que el número de FP y FN debe ser muy pequeño. También, dependiendo del número de VP, VN, FP y FN; podemos evaluar a los algoritmos antes mencionados. A continuación presentamos las mediciones que usaremos en este proyecto, así mismo se discuten sus características y su idoneidad para evaluar.

1) Precisión: Es una medida comúnmente usada en la recuperación de la información para evaluar la calidad de los resultados de búsqueda. Se calcula como: $\frac{VP}{VP+FP}$. Responde a: ¿Cuántos de los etiquetados como correspondientes sí son correspondientes?

La métrica Precisión no incluye el número de VN y no le afecta el desbalanceo de las clases. La precisión calcula la proporción de cuantos de los registros clasificados como "matches" $(VP + FP)$ han sido correctamente clasificados como "matches" verdaderos (VP). Por lo tanto mide el porcentaje de precisión de como un clasificador está clasificando los "matches" verdaderos.

2) Recall: Se calcula como: $\frac{VP}{VP+FN}$. Responde a: De todos los correspondientes, ¿cuántos etiquetamos correctamente?

Recall tampoco incluye el número de VN, por lo que a esta medición tampoco le afecta el problema de desbalance de clases. Recall mide la proporción de "matches" verdaderos $(VP + FN)$ que han sido clasificados correctamente (VP). Por lo tanto, mide cuántos registros que realmente corresponden han sido correctamente clasificados como "matches".

Debe notarse que debe existir una compensación entre Recall y Precisión. Es decir, podría ser importante lograr resultados con un valor alto de Precisión, pero aceptando un valor bajo de Recall, mientras que en otras situaciones valor bajo de Precisión es aceptable pero obligando a tener un Recall alto.

3) Puntaje F1: Calcula la media armónica entre Precisión y Recall. Se calcula, como: $2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$. Es mejor utilizar Puntaje F1 cuando hay algún tipo de equilibrio entre Precisión y Recall en el sistema. Por el contrario, Fi no es tan alta si una medida se mejora a expensas de otra.

Aunque existen más métricas, estas nos van a interesar más debido a la forma de nuestros datos. La Precisión es que tan seguro se está de los Verdaderos Positivos, mientras que Recall es qué tan seguro se está de que no le falta ningún positivo. Escogemos Recall si la idea de falsos positivos es mucho mejor que los falsos negativos, es decir, si la ocurrencia de falsos negativos es inaceptable, cuando se prefiere obtener algunos falsos positivos adicionales en lugar de tener falsos negativos.

Es importante resaltar que hemos realizado la clasificación de nuestras imágenes de dos maneras, en una se engloban las imágenes de manera normal, mientras en otra se aplica un Filtro Gaussiano, además de la obtención de Superpixeles para cada imágen, con el fin de comparar con qué preprocesamiento de las imágenes obtenemos una mejor clasificación.

C. Filtro Gaussiano

El filtro gaussiano se usa para emborronar imágenes y eliminar ruido. Es similar al filtro de media pero se usa una máscara diferente, modelizando la función gaussiana:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2)$$

Cabe mencionar que el filtro gaussiano produce un suavizado más uniforme que otros filtros.

D. Superpixeles

Los superpixeles agrupan pixeles conectados con características similares en regiones perceptualmente significativas. Además, toman ventaja de la redundancia de información para disminuir la dimensión espacial de una imagen y permiten reducir el costo computacional de los algoritmos y mejorar el desempeño en procesamientos posteriores, De esta manera podemos alterar los resultados obtenidos mediante los diferentes algoritmos de clasificación y comparar al proceso que más se ajuste.

IV. Desarrollo

Antes de inicar con el desarrollo del proyecto, mostramos algunas de las bibliotecas (otras las importaremos conforme las vayamos a utilizar) que usaremos para este proyecto:

```
import numpy as np
import pandas as pd
import shutil
import cv2
# paths from imutils atraviesa un directorio y nos permite enumerar todas las imagenes en un directorio.
from imutils import paths
import matplotlib.pyplot as plt
import os
import skimage
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.svm import SVC
from scipy import ndimage
from skimage.segmentation import slic
```

A. Clasifiación: Rayos X

Nosotros no somos capaces de distinguir entre dos imágenes de rayos X y decir qué imagen de rayos X tiene coronavirus o no; sin embargo, una máquina sí puede. Es por esta razón que vamos a hacer varios modelos que puedan predecir si la imagen de Rayos X contiene coronavirus o no.

Primero, recopilamos las imágenes de Rayos X de resultados positivos de paciente con coronavirus.

Kaggle contiene imágenes de rayos X de neumonía, COVID-19, pacientes normales, etc[1]. Por lo tanto, para conseguir las imágenes de rayos X de coronavirus, nos fijamos en el *CSV* que contiene a los metadatos de las imágenes, ya que ahí se encuentra una descripción de cada imagen. Para ello descargamos dicho archivo y lo leemos:

[1]Específicamente, están en el siguiente link: https://www.kaggle.com/bachrr/covid-chest-xray

```
metadatos = pd.read_csv('drive/My Drive/Datos/metadatos.csv')
# Eliminamos las columnas con nan's
metadatos.dropna(axis=1,inplace=True)
# Analizamos la columna finding, la cual contiene informacion de las enfermedades que se encuentran en dichas imagenes de rayos X
metadatos.groupby('finding').count()
```

finding	patientid	view	modality	folder	filename	url
ARDS	4	4	4	4	4	4
COVID-19	296	296	296	296	296	296
COVID-19, ARDS	12	12	12	12	12	12
Chlamydophila	2	2	2	2	2	2
E.Coli	4	4	4	4	4	4
Klebsiella	1	1	1	1	1	1
Legionella	2	2	2	2	2	2
No Finding	3	3	3	3	3	3
Pneumocystis	15	15	15	15	15	15
SARS	16	16	16	16	16	16
Streptococcus	17	17	17	17	17	17

A pesar de que según este archivo indica que hay en total hay 372 registros, es decir, 372 imágenes de rayos X con varias enfermedades, como se muestra en el dataframe anterior, la carpeta con las imágenes solo contiene 357, pero esto no afectará en lo absoluto, pues simplemente nos quedamos con las que sí concuerden con dicha carpeta.

Así, extraemos las imágenes de rayos X que resultaron positivas para COVID-19. En la siguiente parte del código obtenemos las radiografías de los pacientes con COVID-19. Para eso, cambiaremos el conjunto de datos y seleccionamos las filas donde el resultado es igual a COVID-19, y la vista debe ser PA (Posterior Anterior).

```
# Seleccionaremos de todas las combinaciones de pacientes 'COVID-19' con vista de rayos X 'PA'
coronavirus = "COVID-19" # Virus que nos interesa buscar en el set de imagenes
x_ray = "PA" # Vista de los Rayos-X

ruta_img = "drive/My Drive/RP_Proy_Img" # Ruta de la carpeta que contiene las imagenes
ruta_img_salida = 'drive/My Drive/Datos/COVID_19' # Carpeta de salida en la que guardaremos las imagenes

# bucle sobre las filas del marco de datos COVID-19
for (i, row) in metadatos.iterrows():
    if row["finding"] != coronavirus or row["view"] != x_ray:
        continue

    filename = row['filename'].split(os.path.sep)[-1]
    filePath = os.path.sep.join([ruta_img, filename])
    shutil.copy2(filePath, ruta_img_salida)
print('Listo! Se ha hecho la descarga')
```

Después de obtener todas las imágenes de rayos X con COVID-19, pudimos observar que las colocamos en otra carpeta. Se tomaron 141 imágenes de rayos X de COVID-19 para los modelos que vamos a presentar.

Por otro lado, para tener imágenes de personas normales (sanas), sustraemos una muestra de 100[2] y las colocamos en una carpeta junto a la carpeta del conjunto de datos del COVID-19. Esto es, en una carpeta que creamos y nombramos como *Datos* se encuentran las carpetas *Normal* y *COVID_19*, las cuales contienen sus correspondientes imágenes.

Procedemos ahora a guardar cada imagen y su etiqueta: **COVID_19** o **Normal**, donde por obvias razones la primera indica si es sobre coronavirus y la segunda si es de un paciente normal. Las imágenes (ya como arreglos de NumPy) y las etiquetas las guardamos en dos listas separadas. Además, modificamos el tamaño de las imágenes, de tal forma que su dimensión fuera de 224x224, estuvieran en *RGB* y escalamos las intensidades de píxeles de cada imagen al rango [0,1].

```
dataset = "drive/My Drive/Datos" # El dataset
args={}
args["dataset"]=dataset

# colocamos las etiquetas a las imagenes

# iPath contiene la ruta de todas las imagenes
iPaths = list(paths.list_images(args["dataset"]))
# lista que contendra todas las imagenes, tanto las
    de covid como las normales
data = []
# lista que contendra todas las etiquetas de las
    imagenes, es decir,
# indica si la imagen es una de covid o si es de una
     normal
labels = []
for iPath in iPaths:
    # dividimos la ruta de las imagenes, para que
    ellas que tienen un torax 'Normal'
    # y para las que tienen un torax con COVID-19
    label = iPath.split(os.path.sep)[-2]
    # Leemos cada imagen de la ruta de imagenes
    image = cv2.imread(iPath)
    # Convertimos las imagenes a RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Redimensionamos las imagenes
    image = cv2.resize(image, (224, 224))
    data.append(image)
    labels.append(label)
#  escalamos las intensidades de pixeles de cada
    imagen al rango [0,1],
# y convertimos ambas listas como un arreglo de
    NumPy
data = np.array(data) / 255.0
labels = np.array(labels)
```

Ya que las hemos guardado con su correspondiente etiqueta para diferenciarlas, en seguida obtenemos el nombre de las imágenes de cada carpeta, por eso indicamos la ruta en la que se encuentran las dos carpetas (Una es de las imágenes de Rayos X del coronavirus y la otra es de Rayos X de personas normales).

```
Cimages =os.listdir("drive/My Drive/Datos/COVID_19")
Nimages = os.listdir("drive/My Drive/Datos/Normal")
```

[2]Obtenidas de: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Dentro de la parte de la recopilación, volvemos a almacenar en una lista nueva, nuestras imágenes, pero en este caso lo haremos aplicando filtro gaussiano, el cual nos proporciona la biblioteca $ndimage$ mediante la función $gaussian\_filter$, posteriormente con la biblioteca $skimage.segmentation$ Segmentamos la imagen usando el agrupamiento k-means en el espacio del color y obtenemos nuestros super pixeles, con el siguiente código:

```
gausian_data = []
for i in range(len(data)):
  image = ndimage.gaussian_filter(data[i]*255, 3)
  segmentos_slic = slic(image, n_segments=400,
    compactness=10, sigma=1, convert2lab=True)
  image = mean_image(image,segmentos_slic)
  gausian_data.append(image)
gausian_data = np.array(gausian_data) / 255.0
```

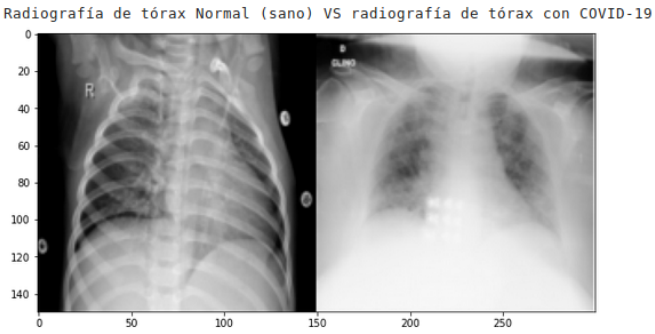
En el cual, se hace llamada a la función $Mean\_image$, y esta realiza el cálculo para la obtención de los superpixeles de cada imágen, tal como se observa en el siguiente código:

```
def mean_image(image,label):
    im_rp = image.reshape((image.shape[0]*image.
    shape[1],image.shape[2]))
    sli_1d = np.reshape(label,-1)
    uni = np.unique(sli_1d)
    uu = np.zeros(im_rp.shape)
    for i in uni:
        loc=np.where(sli_1d==i)[0]
        #print(loc)
        mm=np.mean(im_rp[loc,:],axis=0)
        uu[loc,:]=mm
    img_superpixel = np.reshape(uu,[image.shape[0],
    image.shape[1],image.shape[2]]).astype('uint8')
    return img_superpixel
```

En la siguiente parte, mostramos un par de imágenes de las que hemos recopilado:

```
def grafica(ruta,Cimages,Nimages,i):
  """Funcion que muestra un par de imagenes, la de
    la izquiera contiene una de un torax normal (
    sano),
  mientras que la de la derecha contiene una de un
    torax con COVID-19.
  Parametros:
    ruta(string): Ruta de la carpeta que contiene a
    las dos carpetas: Normal y COVID_19.
    Cimages(list): Nombre de las imagenes de la
    carpeta COVID_19.
    Nimages(list): Nombre de las imagenes de la
    carpeta Normal.
    i(int): Indice de la figura que se quiere
    mostrar
    """

  # leemos la imagen normal que tiene el indice i
  normal = cv2.imread(ruta+"Normal/"+Nimages[i])
  # Cambiamos el tamanio de la imagen para que
    coincida con un cierto tamanio.
  normal = skimage.transform.resize(normal, (150,
    150, 3))
  # leemos la imagen con covid 19 que tiene el
    indice i
  covid = cv2.imread(ruta+"COVID_19/"+Cimages[i])
  # Cambiamos la dimension de la imagen para que
    coincida con un cierto tamanio.
  covid = skimage.transform.resize(covid, (150, 150,
     3) , mode = 'reflect')
  # concatenamos las imagenes anteriores
  parejas = np.concatenate((normal, covid), axis=1)
```

```
    print("Radiografia de torax Normal (sano) VS
        radiografia de torax con COVID-19")
    plt.figure(figsize=(10,5))
    plt.imshow(parejas)
    plt.show()

grafica("drive/My Drive/Datos/",Cimages,Nimages,0)
```



Es importante mencionar que las etiquetas de las imágenes están en forma de cadena, y para los algoritmos de aprendizaje de máquina se requiere que sean numéricas. Por esta razón, categorizamos cada etiqueta con un número:

- **1**, si la imagen es de COVID-19
- **0**, si la imagen es de una persona sana.

```
# categorizamos numericamente cada etiqueta
etiquetas = []
for i in range(len(labels)):
    if labels[i] == 'COVID_19':
        etiquetas.append(1)
    else:
        etiquetas.append(0)
etiquetas = np.array(etiquetas)
```

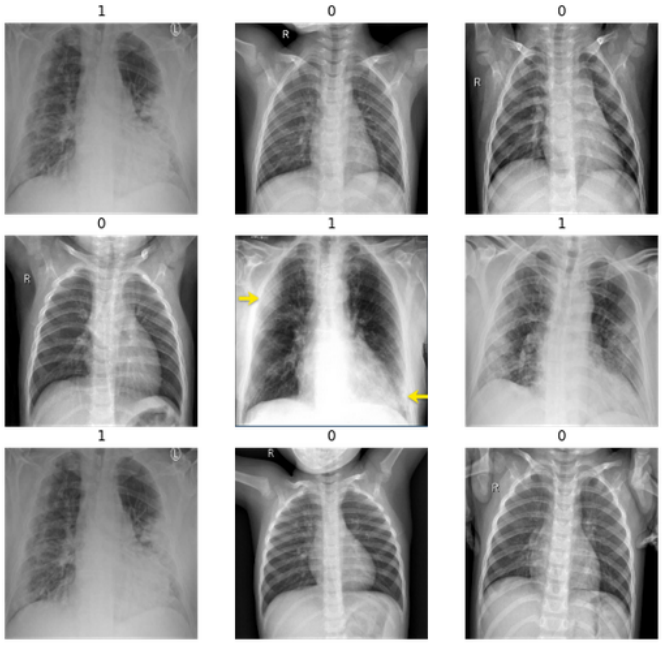
Lo que sigue es dividir nuestros datos en conjuntos de entrenamiento y prueba, para poder usar diferentes modelos de clasificación. Señalamos dicho procedimiento en seguida, y aprovechamos para señalar las dimensiones de cada conjunto:

```
# Dividimos nuestro conjunto de datos en conjuntos
    de entrenamiento y pruebas; donde el conjunto
# de entrenamiento contiene el 80% de los datos,
    mientras que el de prueba contiene el 20%.
(X_train, X_test, Y_train, Y_test) =
    train_test_split(data, etiquetas, random_state
    =42)

# dimensiones de las imagenes de entrenamiento y
    prueba
X_train.shape, X_test.shape, Y_train.shape, Y_test.
    shape

((180, 224, 224, 3), (61, 224, 224, 3), (180,),
    (61,))
```

De esta manera, podemos ver que tenemos 180 imágenes de 224x224x3 para entrenamiento y 61 para prueba.

Encima de esto, visualizaremos algunas figuras (que fueron escogidas aleatoriamente) con su correspondiente etiqueta:

```
fig, axes = plt.subplots(4, 4, figsize = (20, 20))
axes = axes.ravel()
n_training = len(X_train)
for i in np.arange(0, 8):
    # Seleccionamos un indice aleatoriamente
    indice = np.random.randint(0, n_training)
    axes[i].imshow(X_train[indice])
    axes[i].set_title(Y_train[indice])
    axes[i].axis('off')

plt.subplots_adjust(hspace = 0.1)
```

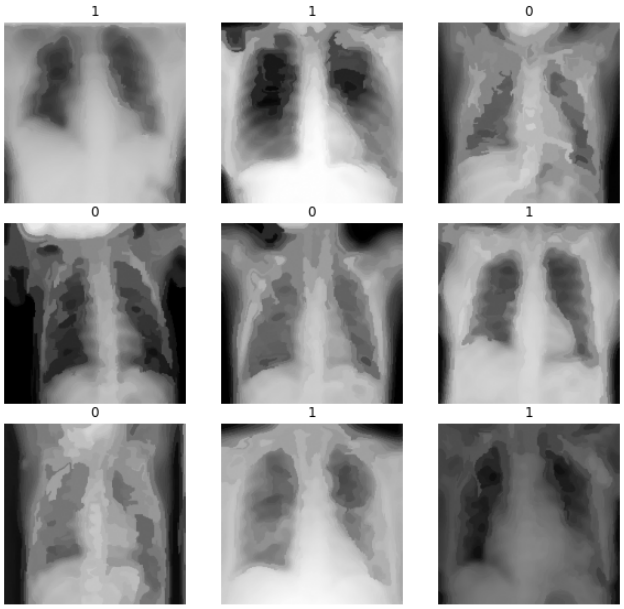


Y realizamos lo mismo para nuestras imágenes modificadas con un filtro gaussiano y sus superpixeles:

```
fig, axes = plt.subplots(3, 3, figsize = (10, 10))
axes = axes.ravel()
n_training = len(X_train_g)
print('Im genes con filtro gaussiano y Superpixeles
    :\n')
for i in np.arange(0, 9):
    indice = np.random.randint(0, n_training)
    axes[i].imshow(X_train_g[indice])
    axes[i].set_title(Y_train_g[indice])
    axes[i].axis('off')

plt.subplots_adjust(hspace = 0.01)
```

Dando los siguientes resultados:

Imágenes con filtro gaussiano y Superpixeles:



Es importante mencionar que para los algoritmos de clasificación, se debe tener en cuenta que los conjuntos que corresponden a la matriz de forma de las características (las X_train, X_test en nuestro código) deben tener una dimensión menor o igual a 2, y anteriormente observamos que eran de 4. Dicho esto, adaptamos esto mediante la función *reshape*:

```
x_train = np.reshape(X_train, (180, X_train.shape[1]
    *X_train.shape[2]*X_train.shape[3]))
x_test = np.reshape(X_test, (61,X_test.shape[1]*
    X_test.shape[1]*X_test.shape[3]))
```

Realizamos este proceso tanto para nuestras imágenes normales, como para nuestras imágenes con filtro:

```
x_train_g = np.reshape(X_train_g, (180, X_train_g.
    shape[1]*X_train_g.shape[2]*X_train_g.shape[3]))
x_test_g = np.reshape(X_test_g, (61,X_test_g.shape
    [1]*X_test_g.shape[1]*X_test_g.shape[3]))
```

Empecemos ahora sí con los modelos de aprendizaje automático. Vamos a clasificar las imágenes comenzando con el algoritmo **KNN** utilizando los 100 vecinos más cercanos:

```
# Creamos el clasificador KNN con 100 vecinos
knn = KNeighborsClassifier(n_neighbors = 100)
# Ajustamos el clasificador a los datos
Y_pred = knn.fit(x_train,Y_train).predict(x_test)
```

Y evaluamos esta primera clasificación:

```
reporte = classification_report(Y_test, Y_pred)
print(reporte)
```

	precision	recall	f1-score	support
0	0.93	0.96	0.95	27
1	0.97	0.94	0.96	34
accuracy			0.95	61
macro avg	0.95	0.95	0.95	61
weighted avg	0.95	0.95	0.95	61

Podemos notar que afortunadamente distingue muy bien cada clase. Para interpretar esta evaluación, vamos a aclarar lo que significan las métricas *precision* y *recall* (La de *f1-score* no la ilustramos porque está relacionado con las otras dos). Estas aclaraciones significarán lo mismo en los modelos posteriores, por lo que en esos casos ya no será necesario volver a repetir la misma idea que hay detrás de estas medidas. Comencemos con la clase 0 (Normal):

- *precision*: El 0.93 indica que el 93% de los registros de este clasificador han sido clasificados como "Normal" y que efectivamente corresponden a "Normal".
- *recall*: El 0.96 significa que se ha clasificado correctamente como "Normal" al 96% de los registros.

En cuanto a la clase 1 (COVID-19):

- *precision*: El 0.97 indica que el 97% de los registros de este clasificador han sido clasificados como "COVID-19" y que efectivamente corresponden a "COVID-19".
- *recall*: El 0.94 significa que se ha clasificado correctamente como "COVID-19" al 94% de los registros.

Realizamos la misma evaluación para nuestras imágenes con filtro gaussiano y superpixeles:

```
knn = KNeighborsClassifier(n_neighbors = 100)
Y_pred_g = knn.fit(x_train_g,Y_train_g).predict(
    x_test_g)
```

Y evaluamos dicha clasificación:

```
reporte = classification_report(Y_test_g, Y_pred_g)
print(reporte)
```

--- Imágenes con filtro Gaussiano y Superpixeles:

	precision	recall	f1-score	support
0	0.77	0.96	0.85	24
1	0.97	0.81	0.88	37
accuracy			0.87	61
macro avg	0.87	0.88	0.87	61
weighted avg	0.89	0.87	0.87	61

Como podemos observar, los resultados obtenidos indican que dicho algoritmo tuvo menor precisión que al utilizar las imágenes sin filtro gaussiano.

Ahora bien, vamos a hacer lo mismo que acabamos de realizar; pero esta vez con el clasificador **Naive Bayes** en sus tres *modalidades*: **Multinomial**, **Bernoulli**, **Gaussiano**; empezando por el primero que mencionamos. Además, imprimiremos la evaluación de dicho modelo:

```
# Ajustamos el clasificador a los datos
# Creamos otro clasificador: Naive Bayes Multinomial
nbm = MultinomialNB()

Y_pred2 = nbm.fit(x_train,Y_train).predict(x_test)

#Generamos el reporte del modelo
```

```
reporte2 = classification_report(Y_test, Y_pred2)
print(reporte2)
```

	precision	recall	f1-score	support
0	0.93	0.96	0.95	27
1	0.97	0.94	0.96	34
accuracy			0.95	61
macro avg	0.95	0.95	0.95	61
weighted avg	0.95	0.95	0.95	61

Notemos que aquí logramos una clasificación igual de buena que con **KNN**.

Ahora realizamos mismo procedimiento, utilizando nuestras imágenes con filtro gaussiano

```
nbm = MultinomialNB()

Y_pred2_g = nbm.fit(x_train_g, Y_train_g).predict(
    x_test_g)

reporte2 = classification_report(Y_test_g, Y_pred2_g
    )
print(reporte2)
```

--- Imágenes con filtro Gaussiano y Superpixeles:

	precision	recall	f1-score	support
0	0.85	0.96	0.90	24
1	0.97	0.89	0.93	37
accuracy			0.92	61
macro avg	0.91	0.93	0.92	61
weighted avg	0.92	0.92	0.92	61

Seguimos con el **Bernoulli**, repitiendo el mismo procedimiento:

```
# Ajustamos el clasificador a los datos
# Creamos otroclasificador_ Naive Bayes Bernoulli
nbb = BernoulliNB()

Y_pred5 = nbb.fit(x_train, Y_train).predict(x_test)

#Generamos el reporte del modelo
reporte5 = classification_report(Y_test, Y_pred5)
print(reporte5)
```

	precision	recall	f1-score	support
0	0.91	0.78	0.84	27
1	0.84	0.94	0.89	34
accuracy			0.87	61
macro avg	0.88	0.86	0.86	61
weighted avg	0.87	0.87	0.87	61

Podemos ver que este clasificador también es bueno, aunque no tanto como los anteriores; pues el valor de las métricas es en general menor al de los pasados.

Si hacemos el modelo para nuestras imágenes con filtro gaussiano y superpixeles, obtenemos:

```
nbm = BernoulliNB()

Y_pred2_g = nbm.fit(x_train_g, Y_train_g).predict(
    x_test_g)

reporte2 = classification_report(Y_test_g, Y_pred2_g
    )
print(reporte2)
```

--- Imágenes con filtro Gaussiano y SuperPixeles:

	precision	recall	f1-score	support
0	0.76	0.54	0.63	24
1	0.75	0.89	0.81	37
accuracy			0.75	61
macro avg	0.76	0.72	0.72	61
weighted avg	0.76	0.75	0.74	61

De igual manera, encontramos una precisión menor.

Continuamos con el **Gaussiano**:

```
# Creamos otroclasificador_ Naive Bayes Gauss
nbg = GaussianNB()

Y_pred3 = nbg.fit(x_train, Y_train).predict(x_test)

#Generamos el reporte del modelo
reporte3 = classification_report(Y_test, Y_pred3)
print(reporte3)
```

	precision	recall	f1-score	support
0	0.96	0.96	0.96	27
1	0.97	0.97	0.97	34
accuracy			0.97	61
macro avg	0.97	0.97	0.97	61
weighted avg	0.97	0.97	0.97	61

Y vemos que de los cuatro algoritmos que hemos utilizados, este último es el que mejor clasifica.

Ahora calculamos con nuestras imágenes con filtro gaussiano para evaluar comportamiento.

```
nbg = GaussianNB()

Y_pred3_g = nbg.fit(x_train_g, Y_train_g).predict(
    x_test_g)

reporte3 = classification_report(Y_test_g, Y_pred3_g
    )
print(reporte3)
```

--- Imágenes con Filtro Gaussiano y Superpixeles:

	precision	recall	f1-score	support
0	0.96	0.96	0.96	24
1	0.97	0.97	0.97	37
accuracy			0.97	61
macro avg	0.97	0.97	0.97	61
weighted avg	0.97	0.97	0.97	61

De igual manera, observamos que no se encontraron alteraciones y muestra ser uno de los algoritmos más precisos.

Asi mismo, tamnbién vale la pena realizar este proceso mediante **SVM**:

```
# Creamos otroclasificador SVM
svm = SVC()

Y_pred4 = svm.fit(x_train,Y_train).predict(x_test)

#Generamos el reporte
reporte4 = classification_report(Y_test, Y_pred4)
print(reporte4)
```

	precision	recall	f1-score	support
0	0.96	0.96	0.96	27
1	0.97	0.97	0.97	34
accuracy			0.97	61
macro avg	0.97	0.97	0.97	61
weighted avg	0.97	0.97	0.97	61

Y con este alcanzamos las mismas evaluaciones que con el clasificador Naive Bayes Gaussiano.

Veamos que sucede con las imágenes con filtro gaussiano.

```
svm = SVC()

Y_pred4_g = svm.fit(x_train_g,Y_train_g).predict(
    x_test_g)

#Generamos el reporte
reporte4 = classification_report(Y_test_g, Y_pred4_g
    )
print(reporte4)
```

--- Imágenes con Filtro Gaussiano y Superpixeles

	precision	recall	f1-score	support
0	0.96	0.96	0.96	24
1	0.97	0.97	0.97	37
accuracy			0.97	61
macro avg	0.97	0.97	0.97	61
weighted avg	0.97	0.97	0.97	61

Y de igual manera encontramos la misma precisión.

Sin embargo, puesto que las métricas indican que la clasificación es mejor cuando no usamos filtro gaussiano ni superpíxeles, decidimos quedarnos con los modelos que no usan dicho pre-procesamiento de las imágenes; pues probablemente este rendimiento se deba a que las imágenes, al ser de rayos X, requieren mayor precisión en los píxeles, y al usar superpíxeles y el filtro gaussiano, podríamos perder información relevante y que no necesariamente es redundante.

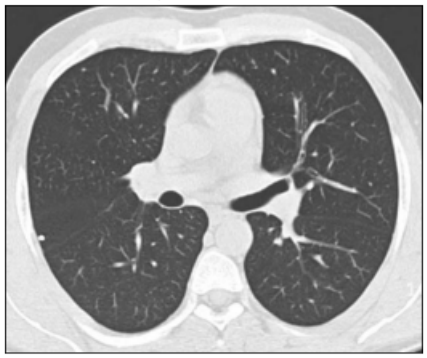
B. Tomografías Axiales pulmonares

Además del análisis realizado con radiografías, decidimos realizar una clasificación a partir de imágenes obtenidas de Medical segmentation. Aquí se presentaron imágenes de pulmones que sufrían daños a causa del COVID-19.

TC axial pulmonar con COVID

Para poder crear un clasificador, necesitábamos de imágenes de pulmones que no tuviesen el mismo tipo de daño, es decir, aquellos que no presentaran COVID-19. Es importante mencionar que esto no quiere decir que el pulmón esté sano. Fue complicado encontrar imágenes pulmonares que cumplieran con estos requisitos, sin embargo, gracias a una buena búsqueda en google se consiguió obtener cierto número de imágenes que cumplieran con nuestros requisitos. Algunas que encontramos son como la siguiente:

TC axial pulmonar sin COVID



Como es de esperar, las imágenes podrían variar en tamaños, por lo que decidimos buscar el menor tamaño y ajustar las imágenes a dicho tamaño. Las imágenes descargadas de Medical segmentation. no tenían (en apariencia) variación, todas eran de 512 x 512. En cambio, las imágenes que se recolectaron manualmente sí presentaron problemas, pues las dimensiones variaban bastante; la imagen con menor dimesión era de 188 x 233.

```
min1 = 100000 # Solo son valores iniciales, no
    representan nada
min2 = 100000 # Solo son valores iniciales, no
    representan nada
for i in range(65):
  aux = train_img.get_fdata()[:,:,i]

  if  aux.shape[0] < min1:
    min1 = aux.shape[0]
  if aux.shape[1]<min2:
    min2 = aux.shape[1]

print(f"La menor dimension es {min2} x {min1}")
>> La menor dimension es 512 x 512


def Encuentra_la_menor_dim(names,ruta):
```

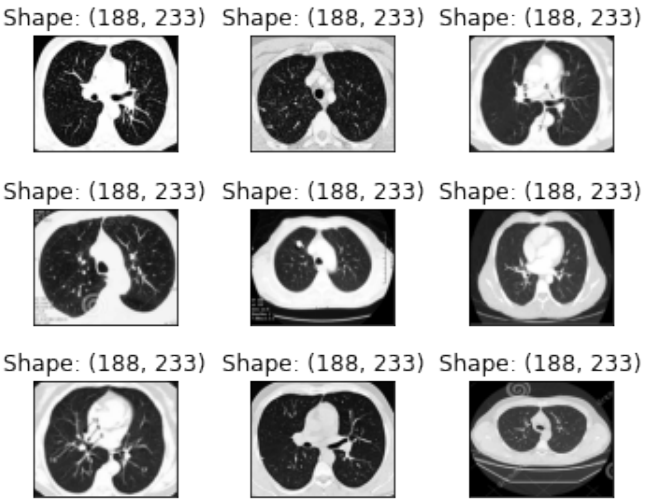
```
    min_C = 10000 # Solo son valores iniciales, no
        representan nada
    min_R = 10000 # Solo son valores iniciales, no
        representan nada

    for i in range(len(names)):
        aux  = io.imread(ruta+names[i],as_gray=True)

        if aux.shape[0] < min_R:
            min_R = aux.shape[0]
        if aux.shape[1] < min_C:
            min_C = aux.shape[1]
    return min_C,min_R

a,b=Encuentra_la_menor_dim(Imagenes,'./
    Imagenes_Sin_covid/')
print(f"La imagen mas pequenia es de {b} x {a}")
>> La imagen mas pequenia es de 188 x 233
```

Una vez que detectamos el mínimo valor de las dimensiones, procedimos a ajustar cada una de las imágenes a este valor.

```
def Redimensiona(imagen,shape):

    imagen = resize(imagen, shape, anti_aliasing=
    True)

    plt.title("Shape: "+str(imagen.shape))
    plt.imshow(imagen,cmap='gray')
    plt.xticks([])
    plt.yticks([])
```

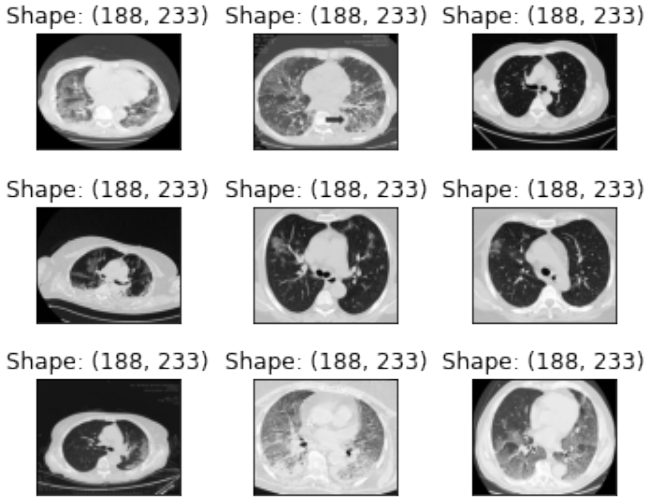
Comenzamos mostrando las imágenes **sin COVID-19** redimensionadas.

```
# Trabajando con las imagenes sin COVID
# Mostraremos solo 10
plt.figure(figsize=(10,4))
for i in range(10):
  img = io.imread('./Imagenes_Sin_covid/'+Imagenes[i
    ],as_gray=True)
  plt.subplot(2,5,i+1)
  Redimensiona(img,(b,a))
plt.tight_layout()
plt.show()
```



Posteriormente mostramos las imágenes con COVID-19 ya redimensionadas.

```
plt.figure(figsize=(10,4))
#Mostrando las 10 primeras
for i in range(10):
  img = train_img.get_fdata()[:,:,i].T
  plt.subplot(2,5,i+1)
  Redimensiona(img,(b,a))
plt.tight_layout()
plt.show()
```



Una vez que comprobamos el rendimiento de redimensionar las imágenes, procedemos a agruparlas.

```
Sin_Cov = []
for imagen in Imagenes:
  img = io.imread('./Imagenes_Sin_covid/'+imagen,as
    _gray=True)
  img = resize(img, (b,a), anti_aliasing=True)
  img = img.reshape(1,-1)
  Sin_Cov.append(img)

Con_Cov = []
for imagen in range(len(Imagenes)):
  img = train_img.get_fdata()[:,:,i].T
  img = resize(img, (b,a), anti_aliasing=True)
  img = img.reshape(1,-1)
  Con_Cov.append(img)
```

Llegados a este punto contamos con 2 conjuntos de imágenes, aquellas que tienen COVID-19 y aquellas que no. Agruparemos estos 2 conjuntos en uno solo (X), donde aplanaremos cada imagen a un vector de (1, (188x233)), y, claro, etiquetamos cada uno con su valor correspondiente Las etiquetas son análogas al caso de los rayos-X del tórax:

- **1**, si la imagen es de COVID-19.
- **0**, si la imagen no es de COVID-19.

```
X = []
Y = []
for i in range(len(Imagenes)):
  X.append(Sin_Cov[i][0])
  Y.append(0) # 0 = Sin Covid
for i in range(len(Imagenes)):
  X.append(Con_Cov[i][0])
  Y.append(1) # 1 = Con Covid
```

Para estar seguros que no hay fallas, comparamos las longitudes de los nuevos conjuntos creados.

```
print("Comparando longitudes")
print(f"X = {len(X)}\nY = {len(Y)}")
>> Comparando longitudes
>> X = 128
>> Y = 128
```

Una vez que contamos con nuestros conjuntos X y Y, procedemos a dividirlos para obtener nuestros valores de prueba y entrenamiento que usaremos con cada uno de nuestros clasificadores.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.25, random_state=6)
```

Comenzamos aplicando una clasificación por KNN.

```
from sklearn.neighbors import KNeighborsClassifier

# Creamos el clasificador KNN con 6 vecinos
knn = KNeighborsClassifier(n_neighbors = 6)
Y_pred = knn.fit(X_train,y_train).predict(X_test)
```

Verificamos el rendimiento de nuestro modelo.

```
from sklearn.metrics import classification_report

#KNN
report = classification_report(y_test, Y_pred)
print(report)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
accuracy			1.00	32
macro avg	1.00	1.00	1.00	32
weighted avg	1.00	1.00	1.00	32

No sólo utilizamos KNN. Se probó con diferentes clasificadores para comparar rendimientos. Posterior a KNN, usamos **Bayes**.

```
from sklearn.naive_bayes import GaussianNB

Bayes = GaussianNB()

#Se utilizan los mismos conjuntos generados para KNN
Bayes.fit(X_train,y_train)

Y_pred = Bayes.predict(X_test)
```

Verificando el rendimiento del modelo

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
accuracy			1.00	32
macro avg	1.00	1.00	1.00	32
weighted avg	1.00	1.00	1.00	32

Curiosamente (En la sección de resultados se apreciará mejor) a pesar de tener estos scores de 1 le resulta difícil clasificar las imágenes. La mayoría las clasifica como "Sin Covid".

Por supuesto que no podía faltar K-means dentro de nuestros clasificadores.

```
from sklearn import cluster

kmeans = cluster.KMeans(n_clusters = 2 )
#Igual que en los anteriores, los conjuntos son los
    mismos
kmeans.fit(X_train,y_train)

Y_predict = kmeans.predict(X_test)
```

Verificando el rendimiento.

```
#K_means
Reporte = classification_report(y_test, Y_predict)
print(Reporte)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
accuracy			1.00	32
macro avg	1.00	1.00	1.00	32
weighted avg	1.00	1.00	1.00	32

Realmente no presentó fallas en las veces que clasificamos para comprobar.

Finalmente clasificamos con SVM.

```
from sklearn.svm import SVC

svm = SVC()
svm.fit(X_train,y_train)
Y_pred = svm.predict(X_test)
```

Verificamos el rendimiento

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
accuracy			1.00	32
macro avg	1.00	1.00	1.00	32
weighted avg	1.00	1.00	1.00	32

Nuevamente no tuvimos problemas con el clasificador.

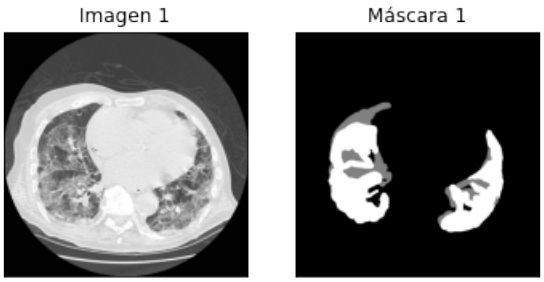
Por lo tanto, como observamos en imágenes anteriores, pudimos notar que todos los clasificadores usados en esta sección, separa adecuadamente cada clase. No obstante, el único que presentó problemas, a pesar de sus scores fue **Bayes**. Fuera de Bayes, todos los clasificadores eran buenos y correspondían a sus scores.

C. Identificando regiones internas

Pensamos en que podríamos identificar algunas regiones en pulmones infectados con COVID-19. Para el análisis usamos Kmeans, KNN, Bayes y SVM.

```
from google.colab import files
files.upload() #Cargamos las imagenes

import nibabel as nib
from matplotlib import pyplot as plt
```

```
import numpy as np

# Cargamos las imagenes
Imagenes = nib.load("./tr_im.nii")
# Cargamos las mascaras
Mascaras = nib.load("./tr_mask.nii")

# Mostramos la imagen 1 y su mascara
n = 0
plt.figure(figsize=(6,6))

plt.subplot(1,2,1)
plt.title("Imagen "+str(n+1))
plt.imshow(Imagenes.get_fdata()[:,:,n].T, cmap='gray')
plt.xticks([])
plt.yticks([])

plt.subplot(1,2,2)
plt.title("Mascara "+str(n+1))
plt.imshow(Mascaras.get_fdata()[:,:,n].T, cmap='gray')
plt.xticks([])
plt.yticks([])

plt.show()
```



Tenemos diferentes regiones acorde a cada imagen, por lo que revisamos las regiones en cada imágen y almacenamos el máximo de categorías. Las regiones las obtuvimos acorde a las máscaras correspondientes a la imagen.

```
max_uniq = 0
for i in range(Mascaras.shape[2]):
  if len(np.unique(Mascaras.get_fdata()[:,:,i])) > max_uniq:
    max_uniq = len(np.unique(Mascaras.get_fdata()[:,:,i])

print(f"El numero maximo de regiones a idenficar es: {max_uniq}")

>> El numero maximo de regiones a idenficar es: 4
```

De lo anterior concluimos que las regiones máximas son 4. Teniendo en cuenta la cantidad de regiones, agrupamos por pixeles pertenecientes a ellos.

```
Regiones = [[] for i in range (max_uniq)]

#Almacenaremos los pixeles de las 100 imagenes de acuerdo a su clasificacion

for i in range(Mascaras.shape[2]):

  # Aplanamos
  img = Imagenes.get_fdata()[:,:,i].reshape(1,-1)
  # El arreglo es [[]] tomamos el primer elemento
  img = img[0]
  #Lo mismo para las mascaras
  mask = Mascaras.get_fdata()[:,:,i].reshape(1,-1)
  mask = mask[0]

  #Analizamos cada pixel de la imagen
  for j in range(len(img)):
    indice = int(mask[j]) # Identificamos la clase a la que pertenece
    Regiones[indice].append(img[j]) # Agregamos el pixel a su clasificaci n
```

Finalmente mostramos la cantidad de pixeles por cada imagen.

```
for i in range(len(Regiones)):
  print(f"La regi n {i+1} cuenta con {len(Regiones[i])} pixeles")
>> La region 1 cuenta con 24394464 pixeles
>> La region 2 cuenta con 1196461 pixeles
>> La region 3 cuenta con 589210 pixeles
>> La region 4 cuenta con 34265 pixeles
```

Teniendo en cuenta la gran diferencia de pixeles entre la región 4 y 1 es de 24,360,199 ajustamos los datos para que tuviesen longitudes iguales, por lo que los igualamos todos al menor (34,265).

```
X = []
Y = []
for i in range(len(Regiones)):
  region = Regiones[i]
  for j in range(34265): # Ajustamos al menor tama o
    X.append(region[j]) # Almacenamos el pixel
    Y.append(i) # Almacenamos su region
X = np.asarray(X)
Y = np.asarray(Y)
```

Una vez que contamos con nuestros datos etiquetados y balanceados, procedemos a separar en entrenamiento y prueba. Para todos los clasificadores anteriormente mencionados, usamos el mismo conjunto de entrenamiento y prueba.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X.reshape(-1,1), Y.reshape(-1,1), test_size=0.30, random_state=6)
```

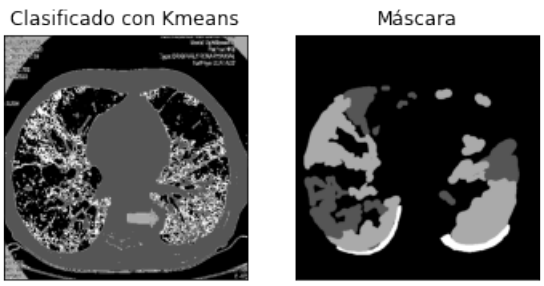
Ahora bien, podemos proceder a aplicar los clasificadores. Empezamos por K-means:

```
from sklearn.cluster  import KMeans

k = 4
Kmeans = KMeans(n_clusters= k )

Kmeans.fit(X_train,y_train)
```

Una vez que nuestro modelo está entrenado, procedemos a realizar una predicción sobre la primera imagen.

```

Clasificado_Kmeans = Kmeans.predict(Imagenes.get_fdata()[:,:,1].reshape(-1,1))

plt.subplot(1,2,1)
plt.title("Clasificado con Kmeans")
plt.imshow(Clasificado_Kmeans.reshape(512,512).T, cmap='gray')
plt.xticks([])
plt.yticks([])
```

```
plt.subplot(1,2,2)
plt.title("Mascara")
plt.imshow(Mascaras.get_fdata()[:,:,1].T,cmap='gray')
plt.xticks([])
plt.yticks([])

plt.show()
```

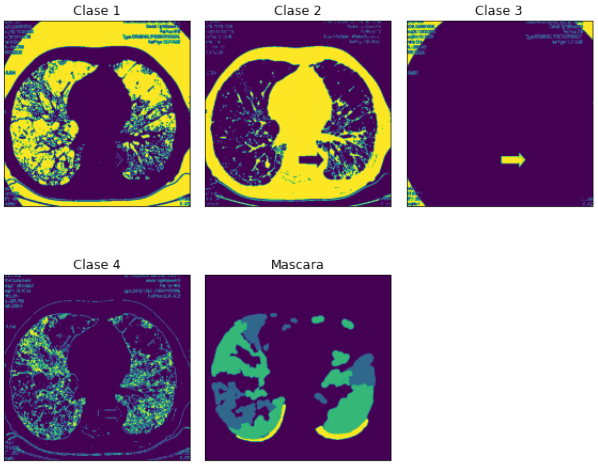


A partir de esto, crearemos una función encargada de dividir la imagen, aislando por categoría.

```
def Obten_mascaras(Imagen):
  # Sera una lista de listas de tamanio igual a los clusters
  mascaras = [[] for i in range(k)]

  # Redimensionamos la imagen
  Imagen = Imagen.reshape(1,-1)[0]

  # Evaluamos cada pixel
  for j in Imagen:

    #Lo agregamos a su grupo
    mascaras[j].append(1)

    #Agregamos un 0 a los demas para mantener la forma
    for i in range(k):
      if j != i:
        mascaras[i].append(0)
  # Convertimos cada mascara a una imagen de 512 x 512
  for i in range(len(mascaras)):
    mascaras[i] = np.asarray(mascaras[i])
    mascaras[i] = mascaras[i].reshape(512,512)

  #Devolvemos la lista de mascaras
  return mascaras
```

Posterior a ello llamamos a la función para observar las máscaras creadas en la imagen anterior. Esto es para ver las clasificaciones con las que contamos y así poder discriminar cuáles sirven y cuáles no.

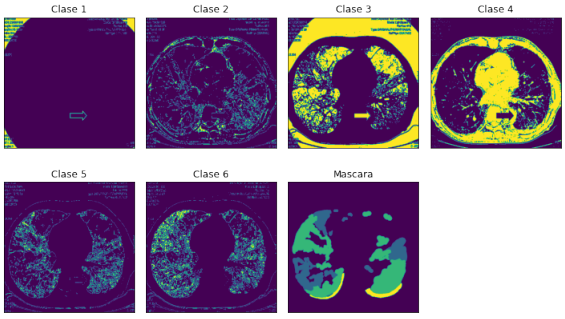
```
MSK = Obten_mascaras(Clasificado_Kmeans.reshape(512,512))

plt.figure(figsize=(10,10))


for i,mascara in enumerate(MSK):
  plt.subplot(5,5,i+1)
  plt.title("Clase "+str(i+1))
  plt.imshow(mascara.T,)
  plt.xticks([])
  plt.yticks([])

plt.subplot(5,5,k+1)
plt.title('Mascara')
plt.imshow(Mascaras.get_fdata()[:,:,1].T)
plt.xticks([])
plt.yticks([])
plt.tight_layout()
plt.show()
```



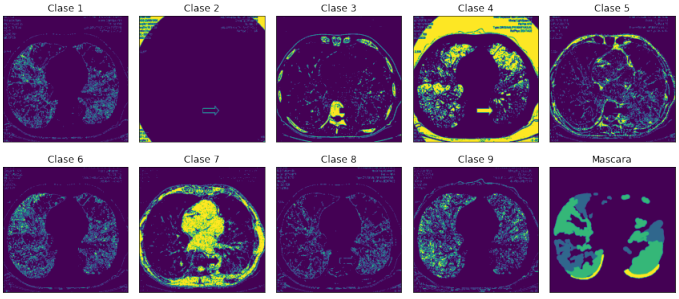
De la imagen anterior podemos darnos cuenta que las clases que más nos interesan son la 1 y la 4. Desgraciadamente, como podemos apreciar, no hubo un clasificador acertivo para derrame pleural.

Se llevo a cabo la calsificación con diferentes K.

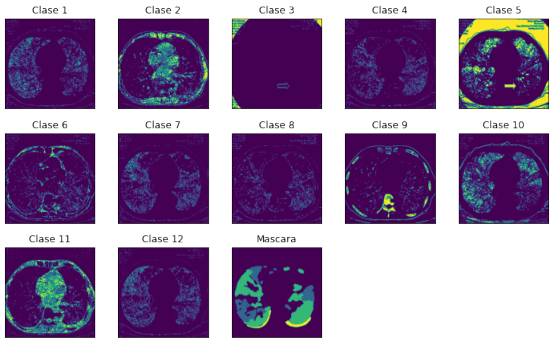
K = 6



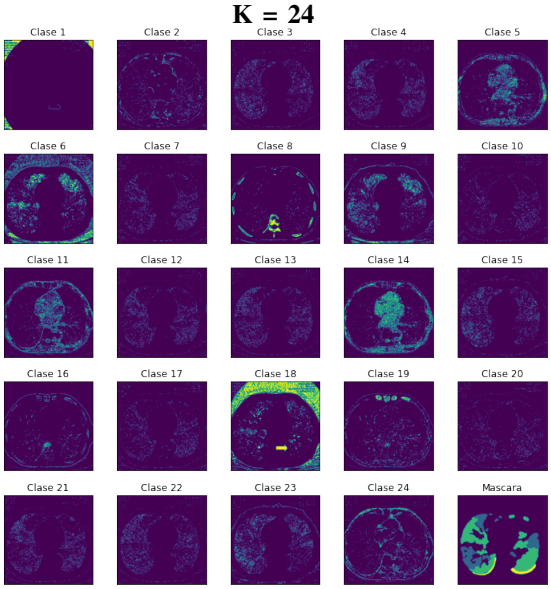
K = 8



K = 12



K = 24



En K =24 y K = 12 tenemos clasificaciones que incluyen el derrame pleural, sin embargo, como podemos apreciar se pierden mucho las clasificaciones de los demás. Por lo que se decidió tomar K = 4.

Para verificar el resultado, clasificamos una imágen al azar y mostramos las 2 clases que sí tenemos.

```
#indica que significa cada clase
Catalogo_Kmeans ={3:"Vidrio Esmeralizado", 1:"
    Consolidaci n"}

import random as rd

def Funcionamiento(Modelo,regiones,catalogo,shape):
  plt.figure(figsize=shape)

  Imagen = rd.randint(0,Imagenes.shape[2])

  prueba = Imagenes.get_fdata()[:,:,Imagen].T

  prueba = prueba.reshape(-1,1)
  Resultado = Modelo.predict(prueba)

  MSK = Obten_mascaras(Resultado.reshape(512,512))
  j=0
  for i,mascara in enumerate(MSK):

    if i in regiones:
      j+=1
      plt.subplot(2,len(regiones),j)
      plt.title(catalogo.get(i))
      plt.imshow(mascara)
      plt.xticks([])
      plt.yticks([])

  plt.subplot(2,len(regiones),len(regiones)+2)
  plt.title('Imagen Original')
  plt.imshow(prueba.reshape(512,512),cmap='gray')
  plt.xticks([])
  plt.yticks([])


  plt.tight_layout()
  plt.show()


Funcionamiento(Kmeans,[1,3],Catalogo_Kmeans,(4,4))
Funcionamiento(Kmeans,[1,3],Catalogo_Kmeans,(4,4))
```

A continuación aplicamos KNN.

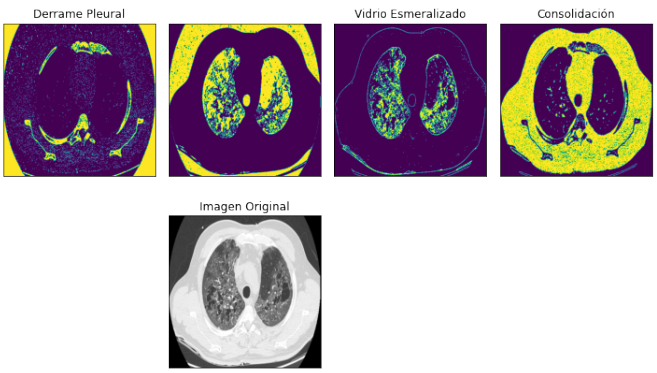
```
from sklearn.neighbors import KNeighborsClassifier

# Creamos el clasificador KNN con 700 vecinos
knn = KNeighborsClassifier(n_neighbors = 700)

Y_pred = knn.fit(X_train,y_train).predict(X_test)
```

A partir de esto verificamos nuestro rendimiento.

```
from sklearn.metrics import classification_report

report = classification_report(y_test, Y_pred)
print(report)
```

	precision	recall	f1-score	support
0	0.90	0.61	0.73	10258
1	0.57	0.66	0.61	10266
2	0.57	0.57	0.57	10297
3	0.71	0.85	0.77	10297
accuracy			0.67	41118
macro avg	0.69	0.67	0.67	41118
weighted avg	0.69	0.67	0.67	41118

Como podemos observar, es bastante bueno para identificar la clase 0. Realmente era bastante tardado (En comparación con Kmeans o Bayes) clasificar una imagen.



Como observamos en la figura anterior, nuevamente tenemos 4 clases. En este caso, cada una de ellas ya está etiquetada. Se puede apreciar la segunda de la primera fila con una falta de etiqueta. En apariencia es el pulmón que no tiene daños, esa no nos interesa por el momento. Otro dato curioso es que dentro de esta clasificación, aparentemente sí tenemos la

posibilidad de identificar el derrame pleural.

Aplicando Bayes.

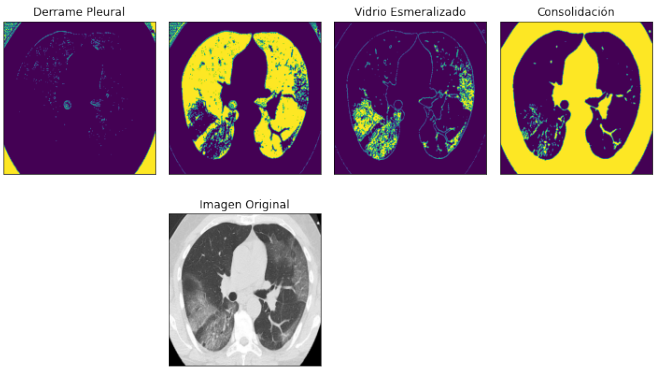
```
from sklearn.naive_bayes import GaussianNB

Bayes = GaussianNB()

Bayes.fit(X_train,y_train)

Y_pred = Bayes.predict(X_test)
```

Verificamos el rendimiento.

```
#Bayes
report = classification_report(y_test, Y_pred)
print(report)
```

	precision	recall	f1-score	support
0	0.98	0.55	0.71	10258
1	0.55	0.75	0.64	10266
2	0.63	0.38	0.47	10297
3	0.62	0.92	0.74	10297
accuracy			0.65	41118
macro avg	0.70	0.65	0.64	41118
weighted avg	0.70	0.65	0.64	41118

De los scores presentados podemos concluir que tiene problemas en algunas regiones.



En la imagen presentada podemos observar que no logra identificar un derrame pleural. Podríamos decir que esto se debe a que no todas las imágenes poseen esta característica. Sin embargo, las demás características las realiza bien. Nuevamente no necesitamos observar el pulmón sano, por lo que la segunda característica (clasificación) queda descartada.

Finalizamos con SVM.

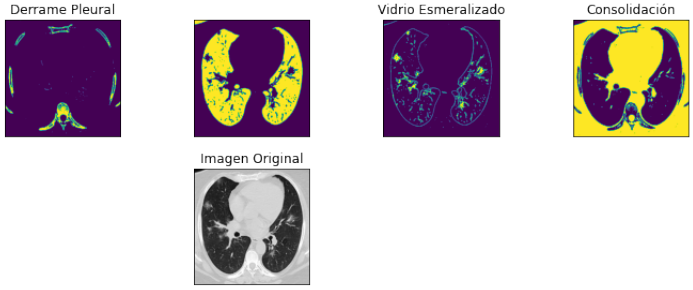
```
from sklearn.svm import SVC

svm = SVC()

svm.fit(X_train,y_train) # tarda bastante

Y_pred = svm.predict(X_test)
```

Verificamos el rendimiento

```
#SVM
Reporte = classification_report(y_test, Y_pred)
print(Reporte)
```

	precision	recall	f1-score	support
0	0.97	0.57	0.72	10258
1	0.58	0.67	0.62	10266
2	0.59	0.58	0.59	10297
3	0.69	0.88	0.77	10297
accuracy			0.68	41118
macro avg	0.71	0.68	0.67	41118
weighted avg	0.71	0.68	0.67	41118

Nuevamente tiene un score bastante regular. Procedemos a ver las clases.



Cabe mencionar que el tiempo de predicción de SVM ess bastante tardado.

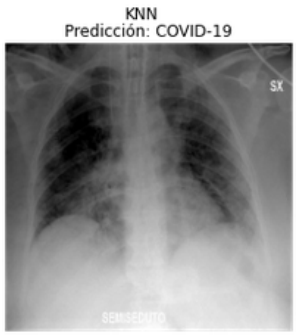
V. Resultados

A. *Rayos X*

Para demostrar qué tan buenos fueron nuestros modelos, a continuación vamos a predecir con cada uno de estos dos imágenes: una de un paciente con coronavirus y otra de un paciente normal. Realizamos dichas predcciones en el mismo orden en que presentamos los algoritmos tanto para nuestras imágenes sin filtros ni superpixeles, como para nuestras imágenes con filtro gaussiano y superpixeles: **KNN**, **Multinomial Naive Bayes**, **Bernoulli Naive Bayes**, **Gaussiano Naive Bayes** y **SVM**.

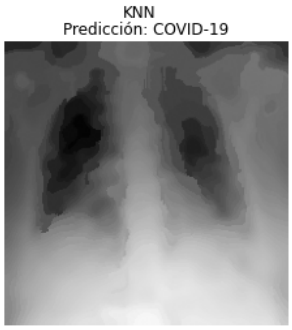
Iniciamos con la imagen de rayos X del tórax de una persona con covid 19:

```
img = cv2.imread('drive/My Drive/covid-19-caso
    -85-5-4.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Redimensionamos la imagen
img = cv2.resize(img, (224, 224))
img = img/ 255.0
img_reshape =  np.reshape(img, (1, img.shape[0]*img.
    shape[1]*img.shape[2]))
clase_knn = knn.predict(img_reshape)
if clase_knn==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('KNN')
plt.imshow(img)
plt.axis('off')
```

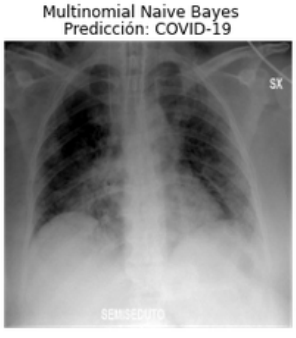



```
image = ndimage.gaussian_filter(img*255, 3)
segmentos_slic = slic(image, n_segments=400,
    compactness=10, sigma=1, convert2lab=True)
image = mean_image(image,segmentos_slic)
image = np.array(image) / 255.0
image_reshape =  np.reshape(image, (1, image.shape
    [0]*image.shape[1]*image.shape[2]))
clase_knn = knn.predict(image_reshape)
print('Imagen con Filtro gaussiano y superpixeles')
if clase_knn==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('KNN')
plt.imshow(image)
plt.axis('off')
plt.show()
```

Imagen con Filtro gaussiano y superpixeles

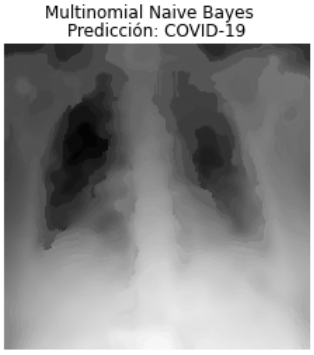



```
clase_nbm = nbm.predict(img_reshape)
if clase_nbm==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Multinomial Naive Bayes')
plt.imshow(img)
plt.axis('off')
```




```
clase_nbm = nbm.predict(image_reshape)
print('Imagen con filtro gaussiano y superpixeles')
if clase_nbm==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Multinomial Naive Bayes')
plt.imshow(image)
plt.axis('off')
plt.show()
```

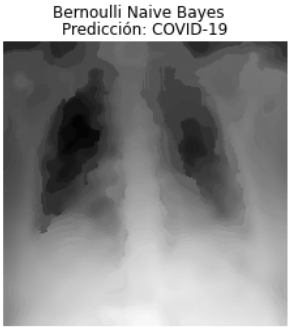
Imagen con filtro gaussiano y superpixeles



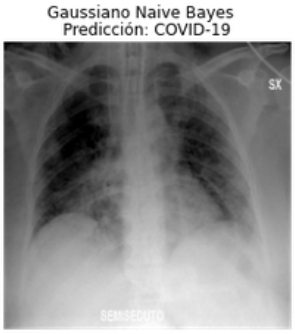

```
clase_nbb = nbb.predict(img_reshape)
if clase_nbb==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Beronoulli Naive Bayes')
plt.imshow(img)
plt.axis('off')
```


```
clase_nbb = nbb.predict(image_reshape)
print('Imagen con filtro gaussiano y superpixeles')
if clase_nbb==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Bernoulli Naive Bayes')
plt.imshow(image)
plt.axis('off')
plt.show()
```

```
Imagen con filtro gaussiano y superpixeles
```

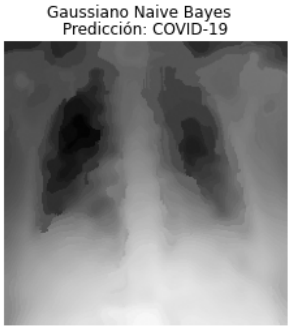



```
clase_nbg = nbg.predict(img_reshape)
if clase_nbg==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Gaussiano Naive Bayes')
plt.imshow(img)
plt.axis('off')
```

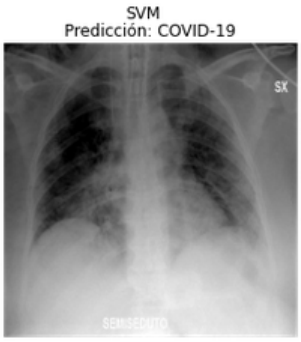



```
clase_nbg = nbg.predict(image_reshape)
print('Imagen con filtro gaussiano y superpixeles')
if clase_nbg==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Gaussiano Naive Bayes')
plt.imshow(image)
plt.axis('off')
plt.show()
```

```
Imagen con filtro gaussiano y superpixeles
```




```
lase_svm = svm.predict(img_reshape)
if clase_svm==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('SVM')
plt.imshow(img)
plt.axis('off')
```




```
clase_svm = svm.predict(image_reshape)
print('Imagen con filtro gaussiano y superpixeles')
if clase_svm==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('SVM')
plt.imshow(image)
plt.axis('off')
plt.show()
```

```
Imagen con filtro gaussiano y superpixeles
```

Y en las figuras anteriores verificamos lo bien que clasifican los 5 modelos usados; pues dada una imagen de una persona que nosotros sabíamos que padecía coronavirus, predijo que efectivamente era de una persona con coronavirus. Aunque las imágenes sin filtros gaussianos ni superpíxeles tenían un mejor rendimiento en los algoritmos, al menos en esta parte las dos formas de pre-procesar las imágenes logra una correcta predicción.

Posteriormente, continuamos con la imagen de rayos X del tórax de una persona normal, de igual manera tanto para la imagen con filtro gaussiano y superpixeles, como para la imagen sin ninguna de estas dos cosas:

```
img2 = cv2.imread('drive/My Drive/
    chest_xray_test_NORMAL_IM-0003-0001.jpeg') #
    insert a random normal x-ray image
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
# Redimensionamos la imagen
img2 = cv2.resize(img2, (224, 224))
img2 = img2/ 255.0
img2_reshape =  np.reshape(img2, (1, img2.shape[0]*
    img2.shape[1]*img2.shape[2]))
clase2_knn = knn.predict(img2_reshape)
if clase2_knn==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('KNN')
plt.imshow(img2)
plt.axis('off')
```


```
image2 = ndimage.gaussian_filter(img2*255, 3)
segmentos_slic = slic(image2, n_segments=400,
    compactness=10, sigma=1, convert2lab=True)
image2 = mean_image(image2,segmentos_slic)
image2 = np.array(image2) / 255.0
image2_reshape =  np.reshape(image2, (1, image2.
    shape[0]*image2.shape[1]*image2.shape[2]))
clase2_knn = knn.predict(image2_reshape)
print('Imagen con filtro gaussiano y superpixeles')
if clase2_knn==[0]:
  plt.title('Predicci n: Normal')
else:
  plt.title('Predicci n: COVID-19')
plt.suptitle('KNN')
plt.imshow(image2)
plt.axis('off')
plt.show()
```

```
print('Imagen con filtro gaussiano y superpixeles')
if clase2_nbm==[0]:
  plt.title('Predicci n: Normal')
else:
  plt.title('Predicci n: COVID-19')
plt.suptitle('Multinomial Naive Bayes')
plt.imshow(image2)
plt.axis('off')
plt.show()
```

Imagen con filtro gaussiano y superpixeles

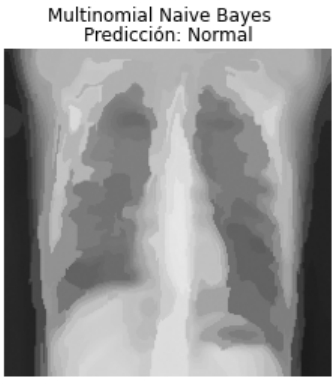
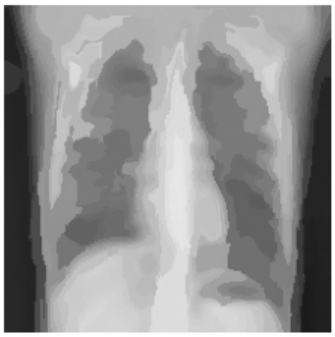


Imagen con filtro gaussiano y superpixeles



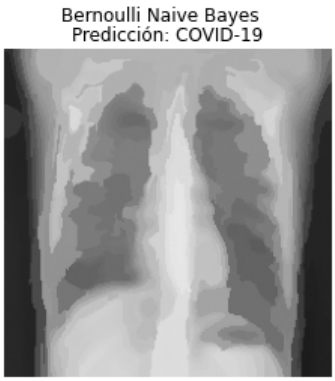

```
clase2_nbb = nbb.predict(img2_reshape)
if clase2_nbb==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Bernoulli Naive Bayes')
plt.imshow(img2)
plt.axis('off')
```

```
clase2_nbm = nbm.predict(img2_reshape)
if clase2_nbm==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Multinomial Naive Bayes')
plt.imshow(img2)
plt.axis('off')
```


```
clase2_nbb = nbb.predict(image2_reshape)
print('Imagen con filtro gaussiano y superpixeles')
if clase2_nbb==[0]:
  plt.title('Predicci n: Normal')
else:
  plt.title('Predicci n: COVID-19')
plt.suptitle('Bernoulli Naive Bayes')
plt.imshow(image2)
plt.axis('off')
plt.show()
```

```
clase2_nbm = nbm.predict(image2_reshape)
```

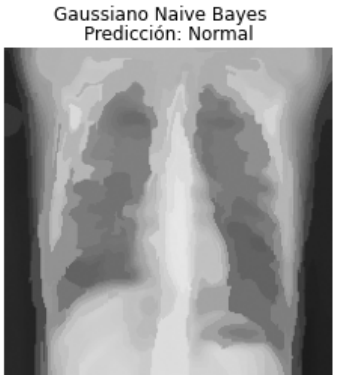
Imagen con filtro gaussiano y superpixeles




```
clase2_nbg = nbg.predict(img2_reshape)
if clase2_nbg==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('Gaussiano Naive Bayes')
plt.imshow(img2)
plt.axis('off')
```


```
clase2_nbg = nbg.predict(image2_reshape)
print('Imagen con filtro gaussiano')
if clase2_nbg==[0]:
  plt.title('Predicci n: Normal')
else:
  plt.title('Predicci n: COVID-19')
plt.suptitle('Gaussiano Naive Bayes')
plt.imshow(image2)
plt.axis('off')
plt.show()
```

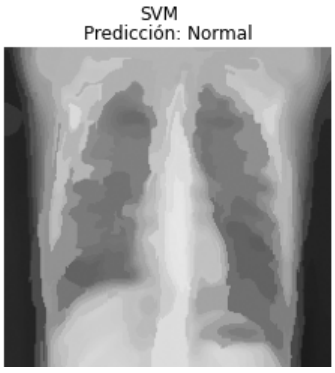
Imagen con filtro gaussiano




```
clase2_svm = svm.predict(img2_reshape)
if clase2_svm==[0]:
  plt.title('Prediccion: Normal')
else:
  plt.title('Prediccion: COVID-19')
plt.suptitle('SVM')
plt.imshow(img2)
plt.axis('off')
```


```
clase2_svm = svm.predict(image2_reshape)
print('Imagen con filtro gaussiano y superpixeles')
if clase2_svm==[0]:
  plt.title('Predicci n: Normal')
else:
  plt.title('Predicci n: COVID-19')
plt.suptitle('SVM')
plt.imshow(image2)
plt.axis('off')
plt.show()
```

Imagen con filtro gaussiano y superpixeles



En las 5 figuras pasadas podemos observar que únicamente con el clasificador Naive Bayes Bernoulli lo hizo incorrectamente para ambos casos, pues una imagen de una persona que nosotros sabíamos que estaba sana (normal), predijo que era de una persona con coronavirus; mientras que en los demás modelos predecían que sí era de una persona normal.

B. Tomografías Axiales pulmonares

Para verificar el rendimiento decidimos seleccionar imágenes de forma aleatoria, pero conociendo su origen, es decir, aún contábamos con los conjuntos separados, solo era necesario un número aleatorio para decidir si elegía pacientes con covid (1) o pacientes sin covid (0). Una vez ahí, seleccionaría aleatoriamente una imagen del conjunto y la clasificaría.

```
import random as rd

Resultados = ({1:"Con Covid",0:"Sin Covid"})

#La presente funci n realiza una predicci n de
    acuerdo al modelo seleccionado para comprobar su
     eficiencia.
```

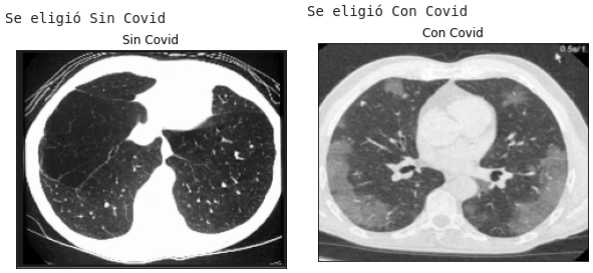
```
def Funcionamiento(modelo):
  Imagen = rd.randint(0,20)
  aleatorio = rd.randint(0,1)

  if aleatorio == 1: # imagen con covid

    prueba = train_img.get_fdata()[:,:,i].T

  else: # Imagen sin covid

    prueba = io.imread('./Imagenes_Sin_covid/'+
    Imagenes[Imagen],as_gray=True)

  prueba = resize(prueba, (b,a), anti_aliasing=True)
  prueba = prueba.reshape(1,-1)
  Resultado = modelo.predict(prueba)

  print("Se eligi ",Resultados.get(aleatorio))

  plt.title(Resultados.get(Resultado[0]))
  plt.imshow(prueba.reshape(b,a),cmap='gray')
  plt.xticks([])
  plt.yticks([])
  plt.show()
```

Iniciamos con **KNN**. Ejecutamos 2 veces la función *Funcionamiento(KNN)* y obtuvimos los siguientes resultados:

Como podemos apreciar en las 2 imágenes anteriores hay 2 líneas de texto por imágen. La primer línea corresponde al conjunto del cual fue tomado. Esto fue únicamente para comprobar que la predicción fuera correcta. La predicción es el título de la imagen (El texto centrado). Coinciden tanto la clasificación como el conjunto del cual fue tomado.

Verificando rendimiento con Bayes. Como mencionamos durante el deasrrollo, a pesar de su score de 1.0, Bayes tenía problemas para clasificar imágenes. Se puede apreciar mejor en las dos imágenes a clasificar, mostradas abajo.

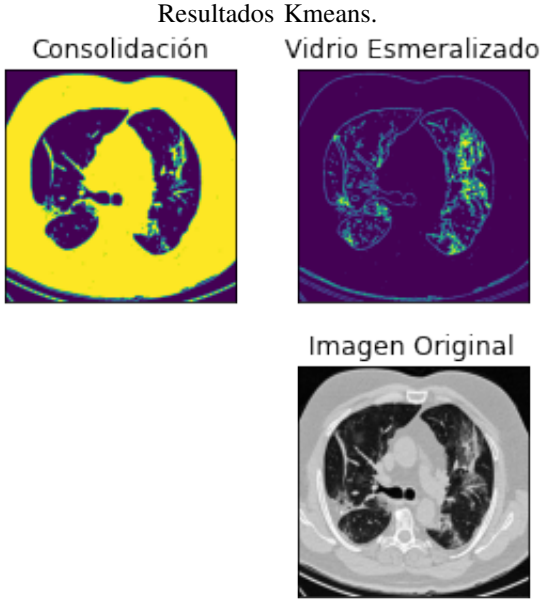
Verificando el rendimiento de Kmeans en las imágenes de abajo. Realmente no tiene problemas.



Verificando el rendimiento para SVM. Curiosamente, en las pruebas realizadas con anterioridad, no aparentaba problemas. Sin embargo, se realizaron nuevas y tuvo un fallo, tal como se puede apreciar en la imagen de abajo a la derecha.

C. Identificando regiones internas

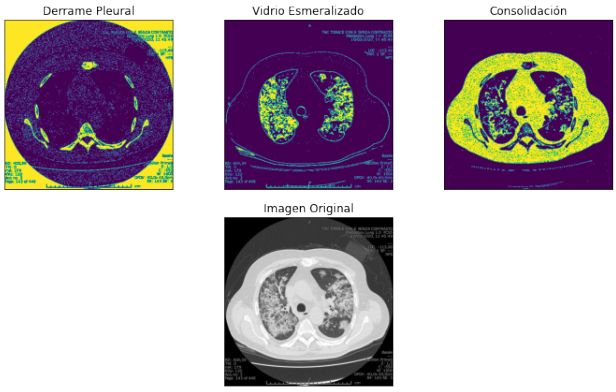
Nota: Los colores de la imágenes se encuentran en escala 'viridis'. Esto es para hacer más fácil de apreciar la zona que ha sido detectada. Mdiante las operaciones vistas con anterioridad podemos saber que el 1 corresponde al pixel que forma parte de la clasificación, por lo que el 0 no corresponderá. En la escala 'viridis' podemos apreciar los colores 1 y 0 de forma realmente contrastante, siento 1 el color amarillo y 0 el color violeta. Por tal motivo podemos concluir que la zona detectada es la amarilla.



El único problema (y que se replica en todos los clasificadores) es que la "Consolidación" No toma únicamente del pulmón,

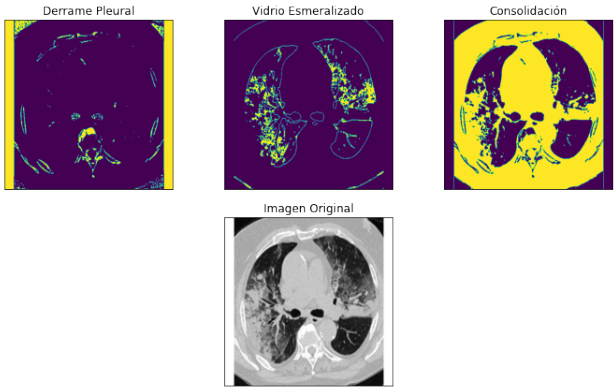
sino que lo fusiona con el perímetro. Sin embargo, si ignoramos el exterior y nos centramos en los pulmones únicamente, podemos apreciar bien lar regiones señaladas.

Resultados KNN.



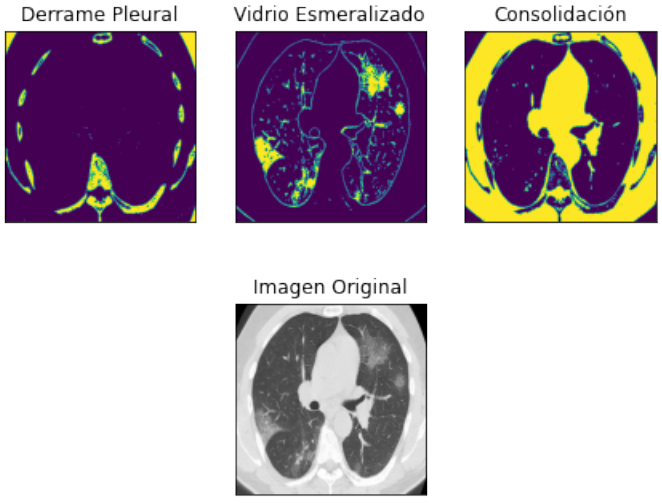
Como podemos apreciar en le imagen anterior, se ubica bastante bien tanto el vidrio esmeralizado como la consolidación. El tiempo de clasificación es relativente lento en comparación con Bayes y Kmeans, pero es aceptable.

Resultados Bayes.



Como podemos apreciar en la imagen anterior, es un buen clasificador. Puede reconocer los 3 valores que necesitamos y su tiempo de respuesta es bastante rápido.

Resultados SVM.



Finalmente contamos con SVM (Imagen anterior), dónde, como se puede apreciar, realiza una buena clasificación. Lo único malo de este método es que tarda demasiado tiempo en generar la clasificación.

VI. Conclusiones

La pandemia del coronavirus ha tomado por sorpresa a todo el mundo, afectando a muchos países, principalmente en lo económico y en cuanto a la salud de las personas. Por ello, a pesar de tomar las correspondiente medidas para evitar la mayor contidad de contagios posibles, se deben implementar herramientas confiables que faciliten la detección de esta enfermedad, y por eso se realizó este proyecto, para comparar el rendimiento de los clasificadores y ver cuál nos podría ayudar de mejor manera a detectar casos potenciales de coronavirus.

Para estas herramientas no se pudieron implementar algunos de los métodos vistos en clases debido a la complejidad de las imágenes. Así mismo, lo que en apariencia podría ser manejado como ”ruido” y limpiado con filtros, no se recomienda llevar a cabo, pues resultaba en características importantes dentro del análisis del esmeralizado o la consolidación, tal como podemos observar al momento de suavizar mediante la aplicación del filtro gaussiano, en algunos algoritmos, llegamos a perder precisión, pues difieren ciertas características que pueden ser de suma importancia al momento de clasificar.

Para el caso de los rayos-X del tórax, pudimos observar que tenemos un buen rendimiento en los algoritmos, aunque la mejora es mayor cuando no realizamos filtro o usamos superpixeles. Además, los algoritmos que mejor puntaje en la evaluación obtuvieron fueron los de Gauss y SVM, por lo que recomendaríamos esos para clasificar futuras imágenes del tórax de las personas, sobretodo si dependiera de cuidar la salud de las personas que padecen esta enfermedad, pues como mencionamos anteriormente, el clasificador Bernoulli no lo hace correctamente para personas que no padecen la enfermedad.

En cuanto a los radiografías de los pulmones, podemos decir que tenemos buenos clasificadores, algunos mejores que otros en tanto logran identificar más áreas, a pesar de que todos tuvieron los mejores puntajes. Sin embargo, todos coinciden en el mismo problema, la consolidación. Resulta que a todos les sucede que la consolidación no sólo toma valores pulmonares, sino que también toma valores del exterior. En apariencia, hablando visualmente no sería un problema, basta con enfocar la atención a los pulmones y las imágenes resaltadas en ellos.

Para identificación de pacientes con COVID a partir de tomografías, observamos que los clasificadores tanto KNN como Kmeans tienen un buen rendimiento para identificar de qué tipo de paciente se trata. Sin embargo, se debería trabajar con más imágenes, 64 por cada conjunto es realmente poco.

Para la identificación de secciones fue un tanto cuanto complicado incluso para nosotros entender cuál era el área consolidada. Como pudimos apreciar, no todos los clasificadores poseen la ”habilidad” para identificar correctamente las regiones. SVM es muy bueno, desgraciadamente el tiempo que tarda en clasificar una sola imagen es realmente tardado. En este caso, Bayes aparenta no tener problemas.

VII. Referencias

- http://bibing.us.es/proyectos/abreproy/11494/fichero/PROYECTO%252FCapitulo+5.pdf
- http://bibing.us.es/proyectos/abreproy/11494/fichero/PROYECTO%252FCapitulo+3.pdf
- https://eprints.ucm.es/56602/1/1138395766-324486_ALEJANDRO_RODR%C3%8DGUEZ_CHAC%C3%93N_Memoria_TFG_2018-2019_3940146_997003508.pdf
- https://cienciadatos.iimas.unam.mx/profesores/pilarang/docencia/calidadyprep/3-1-6-Evaluaciondeclasificacionderegistros.pdf
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- https://rpubs.com/Joaquin$_$AR/233932
- https://rpubs.com/Cristina$_$Gil/SVM
- https://dzone.com/articles/cluster-image-with-k-means
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- http://lapi.fi-p.unam.mx/wp-content/uploads/Superpixeles.pdf