Computación Blanda
Análisis y predicción de contagios Covid-19

Soft Computing
Covid-19 infection prediction analysis

Autor: Ana Manuela Gamboa Piedrahita, Orfilia Castillo Maturana
ISC, Universidad Tecnológica de Pereira, Pereira Colombia
Correo-e: Orfilia.castillo@utp.edu.co
Manuela.gamboa@utp.edu.co

***Resumen*—Este documento presenta un breve análisis planteado para el manejo y predicción de casos de Covid-19 en la región de Risaralda, el Covid-19 es una enfermedad infecciosa causada por el coronavirus que fue descubierta recientemente, normalmente se sabe que esta enfermedad genera cierto tipo de síntomas ya sea como gripa, dolor en el cuerpo, entre otras pero estos síntomas se presentan de manera diferente en cada persona, esta enfermedad fue descubierta en el país de china y se extendió a otros países del mundo considerándose como una pandemia.**
Cuando ésta se expandió llego al país de Colombia el 13 de marzo del 2020 y desde ese punto los casos tuvieron una tendencia de aumento, en la región de Risaralda siempre han estado controlados, pero ha tenido un aumento en los últimos meses, utilizaremos la herramienta de programación Python donde esta cuenta con cierto tipo de librerías y de arreglos matriciales donde es más fácil el análisis de grandes datos; tiene también funciones para dibujar de forma gráfica estos datos y tiene la capacidad de hacer una predicción aproximada de cómo se comportaran los datos insertados.

***Palabras clave*— arreglos, predicción, Python, Machine Learning, librerías.**

Abstract*— *This document presents a brief analysis proposed for the management and prediction of cases of Covid-19 in the region of Risaralda, the Covid-19 is an infectious disease caused by the coronavirus that was discovered recently, normally it is known that this disease generates certain type of symptoms either as flu, pain in the body, among others but these symptoms are presented in a different way in each person, this disease was discovered in the country of China and extended to other countries of the world being considered like a pandemic.
When it expanded it arrived to Colombia on March 13, 2020 and from that point the cases had a tendency of increase, in the region of Risaralda they have always been controlled, but it has had an increase in the last months, we will use the programming tool Python where this has certain type of libraries and matrix arrangements where it is easier the analysis of great data; it also has functions to draw in a graphic way this data and it has the capacity to make an approximate prediction of how the inserted data will behave.

Keywords - arrays, prediction, Python, Machine Learning, libraries.

I. INTRODUCCION

¿Como se propaga?
El virus que causa la COVID-19 se transmite principalmente a través de las gotículas generadas cuando una persona infectada tose, estornuda o espira. Estas gotículas son demasiado pesadas para permanecer suspendidas en el aire y caen rápidamente sobre el suelo o las superficies.
Usted puede infectarse al inhalar el virus si está cerca de una persona con COVID-19 o si, tras tocar una superficie contaminada, se toca los ojos, la nariz o la boca.

uno de los temas más importantes de la actualidad es el covid-19 ya que esta enfermedad es peligrosa, de alto contagio, ha llegado para cambiar la cotidianidad a la que estábamos acostumbrados, y es importante ya que ha cobrado muchas vidas, por esta razón estudiaremos su comportamiento, los casos confirmados, nos enfocaremos en el departamento del Risaralda, y otros datos importantes para cada uno de los municipios.
Usaremos herramientas que nos ofrece el lenguaje Python como Numpy con el cual podremos hacer predicciones utilizando Machine Learning para saber el comportamiento de contagios de la enfermedad y los avances de esta, se explicara la utilización del sistema utilizado y se mostraran los resultados obtenidos del estudio gráficamente.

II. METODOLOGIA

Para la elaboración de este documento se empleó la herramienta de Python como anteriormente se mencionó, para poder hacer la predicción de los datos se realizó con Machine Learning. A continuación, se describirá brevemente las líneas de código utilizadas se describe en categorías en el programa junto con la muestra de datos tomadas para poder hacer la predicción.

Para la toma de los datos nos dirigimos a la página de la gobernación de Risaralda, donde se muestra de manera gráfica y diariamente el número de casos de Covid que se presentan en cada uno de los de municipios del departamento de Risaralda.

MUNICIPIO	CASOS CONFIRMADOS	RECUPERADOS	MUERTES	ACTIVOS
DOSQUEBRADAS	2121	1152	37	930
PEREIRA	5638	2971	130	2537
SANTA ROSA	296	162	10	124
LA VIRGINIA	478	296	5	176
BELEN DE UMBRIA	37	24	2	11
PUEBLO RICO	78	7	1	2
SANTUARIO	34	30	0	4
LA CELIA	10	9	0	1
BALBOA	18	18	0	0
MARSELLA	16	13	1	2
QUINCHIA	28	27	0	1
APIA	11	5	1	5
GUATICA	4	4	0	0
MISTRATO	52	16	0	36
RISARALDA	8821	4802	187	3829

Estas son algunos de los datos tomados de una manera abreviada. Después de tener los datos lo que realizamos es el análisis del código que vamos a utilizar, describiéndolos de la siguiente manera.

1. ENTRENAMIENTO

Se le indica al programa cuales son las librerías que se deben importar, se comenta cual es el funcionamiento de cada una de las librerías y con qué fin se va a utilizar.
import OS; este comando proporciona una forma portátil y mas practica de utilizar la funcionalidad dependiente del sistema operativo. Cuando se esté trabajando si solo se desea el leer o escribir en un archivo se realiza mediante un comando **open ().** También se puede refieren información sobre el entorno de este y nos permiten manipular la estructura de directorios.

Consiguiente a la importación del OS, procedemos a importar los directorios los cuales son DATA_DIR de donde se procedera a sacar la informacion para hacer los calculos de la predicción y por último CHART_DIR en este directorio se guardarán las imagenes de las graficas generadas por el calculo de los datos suministrados en el directorio anterior; se aclara que para poder guardar las imagenes se les debe suministrar un nombre y lo veremos mas adelante. Los directorios se importan de la siguiente manera.

from utils import DATA_DIR, CHART_DIR

import numpy as np

consiguiente a importar los directorios, se procede a importar las librerías de Numpy con un alias, con el fin de llamarlas de una manera abreviada y poder utilizarlas más frecuentemente.
Se eliminan las advertencias por el uso de funciones que en el futuro cambiarán, para poder eliminarlas se realiza con la siguiente instrucción, donde se utilizan las librerías de Numpy.

np. seterr(all='ignore')

Luego se importa la librería **scipy (import scipy as sp)** y después con **(import matplotlib.pyplot as plt)** importamos la librería **matplotlib**.
La librería scipy me permite importar todos los paquetes científicos donde se encuentran los módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen, resolución de ODE y otras tareas para la ciencia e ingeniería. Y Matplotlib es una librería para generar gráficas a partir de datos contenidos en listas, vectores, en el lenguaje de programación Python y en su extensión matemática Numpy.

data = np.genfromtxt(os.path.join(DATA_DIR, "datosCovid.tsv"), delimiter="\t")

Primero se hace le llamado a la instrucción **np.genfromtxt,** esto hace que a partir del texto (es el archivo donde están recolectados los datos) genere información y luego con la instrucción delimiter="\t"), para delimitar y diferenciar un dato de otro dato.
colors=['k', 'y', 'g', 'm', 'r'] linestyles=['-', '-.', '--', ':', '-']
Después de agregar los datos, se definen los colores y los tipos de líneas que llevara las gráficas; los colores que se definieron para este programa fueron: g = verde, k = negro, b = azul, m = magenta, r = rojo y=amarillo

data = np.array(data, dtype=np.float64)
En esta función se le indica a la variable data que establece el tipo de dato con el que estamos trabajando.
Y consiguiente a eso se imprimen los errores o valores que fueron mal ingresados en la base de datos o en el documento tsv, esto se logra mediante la siguiente instrucción.

print("Número de entradas incorrectas:", np.sum(np.isnan(y)))

la función **isnan(vector)** devuelve un vector en el cual los TRUE son valores de tipo **nan**, y los valores FALSE son valores diferentes a **nan** (valores que se introdujeron de forma incorrecta y que no es aceptado en el sistema). Con esta información, este vector permite realizar transformaciones a otros vectores (o al mismo vector), y realizar operaciones como sumar el número de posiciones TRUE, con lo cual se calcula el total de valores tipo nan que se encuentra dentro del arreglo.

Consiguiente a esto se define una función donde se define el modelo, el cual contiene el comportamiento del ajuste con base en un grado polinomial elegido para este y se puede apreciar en las figuras que resultan al ejecutar el programa de lección seis (1), en la gráfica se puede apreciar un conjunto de puntos que son la información suministrada en él documento de datosCovid.tsv, consiguiente a esta gráfica se muestra otra grafica con una línea de grado 1 que es el ajuste polinomial que se le desea realizar a los datos que están expresado en el en forma de un conjunto de punto, para poder apreciar esta línea de grado polinomial se realiza con la siguiente función.

def plot_models(x, y, models, fname, mx=None, ymax=None, xmin=None):

También se pueden realizar diferentes rectas con los ajustes polinomiales necesarias para que se adapten al comportamiento de los datos suministrados.
La función **plt.figure(num=None, figsize=(8, 6))** es la que crea o activa las imágenes existentes al momento de ejecutar el programa proporcionando así, un numero identificador ,el cual determina el ancho y el largo de la figura en este caso sus valores son ocho y seis. La función plt.clf() se encarga de borrar el espacio de la figura.
num = identificador.
figsize: anchura, altura.

La figura plt.clf() aparte de borrar los espacios de la figura, también realiza un gráfico de dispersión de la columna (y) frente a la columna (x) que se conoce como el plano cartesiano, con diferentes tamaños y colores de marcador (tamaño=10) plt.scatter(x, y, s=10), en la variable S es donde se proporciona el grosor que tomara cada una de estos puntos en el gráfico. Consiguiente a esto de le asignas los nombres a cada uno de los laterales que se presentaran a en los resultados mostrados en las gráficas.

plt.title("Contagio de Covid-19 desde el inicio Risaralda ") le pone un título en la base del gráfico.
plt.xlabel("Tiempo") el tiempo el cual se muestra en la base de la grafica
plt.ylabel("Número de contagios") y tiene un título lateral

plt.xticks([w * 7 * 24 for w in range(10)], ['meses %i' % w for w in range(20)])

Con la función anterior se obtiene o establece las ubicaciones de las marcas actuales y las etiquetas del eje x. Los primeros corchetes ([]) se refieren a las marcas en x, los siguientes corchetes ([]) se refieren a las etiquetas.
En el primer corchete se tiene: 1*7*4 + 2*7*4 + ..., hasta completar el total de puntos en el eje horizontal, según el tamaño del vector x. Además, se aprovecha para calcular los valores de w, los cuales se agrupan en paquetes de w*7*4. Esto permiten determinar los valores de w desde 1 hasta 6, indicando con ello que se tiene un poco más de 6 meses
Estos valores se utilizan en el segundo corchete para escribir las etiquetas basadas en estos valores de w. Por tanto, se escriben etiquetas para w desde 1 hasta 5, lo cual constituye los meses analizadas.

Evaluamos el tipo de modelo que ase recibió mediante una condición que dice lo siguiente.
if models:
if mx is None:
mx = np.linspace(0, x[-1], 1000)

En caso de que no se defina ningún valor para mx, este será calculado por la función mx = np.linspace(0, x[-1], 80)) se debe tener en cuenta que esta función se encarga de devolver en un intervalo los números espaciados uniformemente.

NOTA: linspace devuelve números espaciados uniformemente.

La función **zip ()** toma elementos iterables (puede ser cero o más), los agrega en una tupla y los devuelve
Aquí se realiza un ciclo.
Zip (): devuelve un iterador de tuplas basado en los objetos iterables.
Si no pasamos ningún parámetro, zip () devuelve un iterador vacío, si se pasa un único iterable, zip () devuelve un iterador de tuplas con cada tupla que tiene solo un elemento.
Si se pasan varios iterables, zip () devuelve un iterador de tuplas y cada tupla tiene elementos de todos los iterables.
En este ciclo se dice que para cada modelo y estilos se utiliza una paleta de colores.

for model, style, color in zip(models, linestyles, colors):
 # print "Modelo:",model
 # print "Coeffs:",model.coeffs
plt.plot(mx, model(mx), linestyle=style, linewidth=3, c=color)

Nota: For: Es un bucle que repite el bloque de instrucciones un número prederminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.
El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.
en el **For** anterior lo que se hace es recorrer todos los modelos y estilos y se le asigna los colores a las listas que están comprimidas con la función zip, esta función lo que hace es que me comprime estas tres listas en una sola y con el for lo que se realiza es recorrerlas.
con esto se obvia hacer un bucle for para cada uno de los arreglos.

Linestyles: Los estilos de línea simples se pueden definir utilizando las cadenas “sólido", "punteado", "discontinuo" o "dashdot". Se puede lograr un control más refinado proporcionando una tupla de guiones.
Legend es decir la función plt.legend(["d=%i" % m.order for m in models], loc="upper left") crea una leyenda con etiquetas descriptivas para cada serie de datos trazada.

esta nos indica o nos muestra en la gráfica la definición de una leyenda en este caso la leyenda es que significa cada línea que se va a dibujar en el modelo, y también se le indica en que posición se quiere mostrar.

plt.autoscale (tight=True)
auto escalada los valores que se van a mostrar en los ejes para poder auto escalar estos valores se le tiene que decir de cuales ejes y como se quieren mostrar en este caso con la definición de (tight =True) se indica que los valores van a estar agrupados o apretados en los dos ejes.

plt.ylim(ymin=0) Obtiene o establece los límites y de los ejes actuales.

plt.ylim: Estos comandos le permiten hacer zoom o expandir el gráfico o establecer los rangos de los ejes para incluir valores importantes (como el origen).

plt.grid(True, linestyle='-', color='0.75') se cconfigura las líneas de la cuadricula se definen los ejes en los cuales se van a trabajar ejemplo: {'ambos', 'x', 'y'}, opcional. El eje sobre el que aplicar los cambios.
b: bool o Ninguno, opcional. Ya sea para mostrar las líneas de la cuadrícula. Si se proporcionan kwargs, se supone que desea que la cuadrícula esté activada y b se establecerá en True.
Si b es Ninguno y no hay kwargs, esto cambia la visibilidad de las líneas.
kwargs= propiedades de las líneas en 2D
linestyle= puede ser desplazamiento, encendido-apagado-secuencia.

plt.savefig(fname) Guarda la figura actual
name= es una ruta, o un objeto similar a un archivo de Python, o posiblemente algún objeto dependiente del backend. si fname no es una ruta o no tiene extensión, recuerde especificar el formato para asegurarse de que se usa el backend correcto.
plot_models(x,y,None,os.path.join(CHART_DIR,"1400_01_0 1.png"))crea gráficos a partir de modelos de regresión, ya sean estimaciones (como los llamados gráficos de bosque o de puntos) o efectos marginales.

A continuación, se ejecuta el código que genera cuatro modelos diferentes, llamados f1, f2, f3, f10 y f100 donde cada una se refiere a una función polinomial que se verán reflejadas en las gráficas. Para la poder realizar el ajuste de los grados polinomiales se da con las siguientes líneas a de código.

fp1, res1, rank1, sv1, rcond1 = np.polyfit(x, y, 1, full=True)
print("Parámetros del modelo fp1: %s" % fp1)
print("Error del modelo fp1:", res1) f1 = sp.poly1d(fp1)

la función anterior presentada es solamente para la función polinomial de grado 1, pero esta instrucción se realiza también para los demás grados y se presentan de la siguiente manera.
Ajuste polinomial de segundo grado:

fp2, res2, rank2, sv2, rcond2 = np.polyfit(x, y, 2, full=True)
print("Parámetros del modelo fp2: %s" % fp2)
print("Error del modelo fp2:", res2) f2 = sp.poly1d(fp2)

los siguientes a presentar son los ajustes polinomiales de grado 3 hasta el centésimo, pero también se pueden agregar mas ajustes polinomiales.
f3 = sp.poly1d(np.polyfit(x, y, 3))
f10 = sp.poly1d(np.polyfit(x, y, 10))
f100 = sp.poly1d(np.polyfit(x, y, 100))

Consiguiente a crear los ajustes polinomiales se realiza la creación de las líneas de código que me permiten presenciar esos ajustes polinomiales dentro de las gráficas; para ver las graficas se deben guardar con un nombre establecido anteriormente y se guardaran en la carpeta CHARTS que se menciona anteriormente.
Las graficas se crean a partir de modelos de regresión ya sean estimaciones (como los llamados gráficos de bosque o de puntos) o efectos marginales

Se agrega los modelos de las imágenes (f1,f2,f3,f10 y f100)
Con las siguientes instrucciones:
plot_models(x,y,[f1],os.path.join(CHART_DIR,"1400_01_ 02. png"))
plot_models(x,y,[f1,f2],os.path.join(CHART_DIR,"1400_0 1_ 03.png"))
plot_models(x,y,[f1,f2,f3,f10,f100],os.path.join(CHART_D IR , "1400_01_04.png"))

Después de crear las graficas tomamos un modelo de los que fueron creados para realizarles un ajuste y dibujándolo conociendo el punto de inflexión.
El punto de inflexión es aquel donde se nota un cambio de los datos mostrados en la gráfica, ya sea un punto donde se aprecia que los datos pueden aumentan o disminuyen, este análisis con el punto de inflexión facilita la predicción de en donde pueden los datos cambiar después del mes 6. El modelo de inflexión se tomo de la siguiente manera.

inflexión = 5 * 7 * 4

se le saca los puntos de inflexión al modelo A, en sus respectivos ejes
xa = x[:int(inflexion)]
ya = y[:int(inflexion)]
se le saca los puntos de inflexion al modelo B, en sus respectivos ejes
xb = x[int(inflexion):]
yb = y[int(inflexion):]

Después de sacar los puntos de los modelos A y B se grafican dos líneas rectas que se ajustan a dos conjuntos de datos que se dan de la siguiente manera:

fa = sp.poly1d(np.polyfit(xa, ya, 1))
fb = sp.poly1d(np.polyfit(xb, yb, 1))

las dos líneas anteriores se pueden evidenciar en la siguiente grafica que se guarda de la siguiente manera.

plot_models(x, y, [fa, fb], os.path.join(CHART_DIR, "1400_01_05.png"))

Como se necesita evaluar si el modelo está bien entonces se reúnen las diferencias entre lo que predice el modelo para un mes determinado y lo que realmente sucedió a ese día. Cuanto mayor sea esta puntuación, peor será el modelo. Y esto se logra con las siguientes instrucciones: donde definimos a error el cual contendrá **f** que es la función; **x,y** que son las columnas de datos que se están analizando.

def error(f, x, y): return np.sum((f(x) - y) ** 2)

siguiendo lo anterior lo que hacemos es mostrar el conjunto de errores que se dan en los datos.
print("Errores para el conjunto completo de datos:")
for f in [f1, f2, f3, f10, f100]:
print("Error d=%i: %f" % (f.order, error(f, x, y)))

Se debe de mostrar los errores para cada una de las funciones f**1, f2, f3, f10, f100**, pero solo cuando se haya pasado el punto de inflexión.

print("Errores solamente después del punto de inflexión")
for f in [f1, f2, f3, f10, f100]:
print("Error d=%i: %f" % (f.order, error(f, xb, yb)))

luego de evidenciar los tipos de errores, se debe crear los gráficos a partir de modelos de regresión, ya sean estimaciones (como los llamados gráficos de bosque o de puntos) o efectos marginales. **os.path** se puede usar para analizar cadenas que representan nombres de archivo en sus partes componentes. Estas funciones operan únicamente en las cadenas.

plot_models(
x, y, [f1, f2, f3, f10, f100],
os.path.join(CHART_DIR, "1400_01_06.png")

Se utiliza la función linspace, esto sirve para generar un array Numpy formado por n números equiespaciados entre dos dados uniformemente durante un intervalo especificado. Devuelve el número de muestras espaciadas uniformemente, calculadas sobre el intervalo (inicio, parada). Su sintaxis es: np.linspace(valor-inicial, valor-final, número de valores).

mx=np.linspace(0 * 7 * 4, 8 * 7 * 4, 10000),
ymax=10000, xmin=0 * 7 * 4) los valores que se muestran dentro de la función como el **7** equivale al número de días que se encuentran en una semana el **4** equivale al número de semanas presentes en un mes y por ultimo el **8** que equivale al numero de meses en el cual se evidencia los contagios de covid-19.

Se deben se construir funciones para los modelos de orden superior que serían (órdenes 2, 3, 10 y 100). Se les asignan a las variables fb2,fb3,fb10,fb100 un ajuste de grado polinomial donde a la función polyfit se le manda con los puntos x, y de la matriz y también recibe el grado polinomial. en las siguientes líneas de código.
print("Entrenamiento de datos únicamente después del punto de inflexión")
fb1 = fb
fb2 = sp.poly1d(np.polyfit(xb, yb, 2))
fb3 = sp.poly1d(np.polyfit(xb, yb, 3))
fb10 = sp.poly1d(np.polyfit(xb, yb, 10))
fb100 = sp.poly1d(np.polyfit(xb, yb, 100))

sp.poly1d es una función de clase polinomial unidimensional. (np.polyfit()) se encarga de ajustar un polinomio de grado deg a los puntos (x, y) .Devuelve un vector de coeficientes p que minimiza el error al cuadrado en el orden deg además este método de clase se recomienda para código nuevo ya que es más estable numéricamente.

En el siguiente for se realiza es un análisis de cada uno de los vectores que se devuelven con los ajustes polinomiales y con el análisis de la reducción de errores, de cada uno de los polinomios. for f in [fb1, fb2, fb3, fb10, fb100]: Luego se muestra en pantalla cada uno de los errores encontrados en los en cada vector; los errores se muestran en el orden que fueron hallados.
print("Error d=%i: %f" % (f.order, error(f, xb, yb))) Se debe de volver a graficar, estos gráficos son después de haber alcanzado el punto de inflexión.

plot_models(x, y, [fb1, fb2, fb3, fb10, fb100], os.path.join(CHART_DIR, "1400_01_07.png"), mx=np.linspace(0 * 7, 11 * 7, 100), ymax=0, xmin=0 * 7)

Esta parte lo que me permite realizar separaran los datos de prueba que se van a entrenar; además se desea evaluar el rendimiento de predicción de un modelo, se debe probar con algunos datos que nunca ha visto. Entonces se le pasa un conjunto de puntos para estimar el modelo: llamamos a este conjunto de puntos datos de entrenamiento. Pero no dejamos que el modelo vea todos nuestros datos. Tenemos que tomar algunos datos para la prueba, y ese conjunto se llama datos de prueba. Esta es una de las ideas clave del paradigma del aprendizaje automático. Entrenamiento y prueba separados; no permita que el modelo en entrenamiento tenga acceso a información sobre los datos de prueba mientras se está entrenando debido a que esto podría generar errores cuando ya se están haciendo la predicción con los verdaderos datos.

frac = 0.3 Esta línea indica que solo se van a utilizar la tercera parte de los datos.

split_idx = int(frac * len(xb)) Aquí se saca el muestreo o el número de elementos que se va a utilizar, entonces se multiplica la fracción 0.3 por la longitud que sería de 57, solo se tiene en cuanta la parte entera.

Consiguiente a esto lo que se realiza son los modelos cuadráticos.

2. MODELO CUADRATICO

En el modelo cuadratico se realiza el analisis de los modelos cuadraticos generadas que se graficaran despues de hacer el analisis.

shuffled = sp.random.permutation(list(range(len(xb))))

Permutación aleatoriamente una secuencia o devolver un rango permutado. Si x es una matriz multidimensional, solo se baraja a lo largo de su primer índice.
print("\n\nshuffled=sp.random.permutation(list(range(len(x b)))) = ", sp.random.permutation(list(range(len(xb)))), "\n\n")
Sirve para ver esa lista barajada de forma aleatoria con la función shuffled.
Consiguiente a esto lo que realizamos es la toma de la matriz permutada y utilizamos la función **sorted**: función ordena los elementos de un iterable dado en un orden específico y devuelve el iterable ordenado como una lista y lo apreciamos de la siguiente manera

test = sorted(shuffled[:split_idx])
train = sorted(shuffled[split_idx:])

de esta manero lo que me retorna la function sorted se guarda en dos listas vacias con el nombre de test y train.

Se vuelve y se realiza un ajuste polinomial con los valores que se guardaron en las variables test, train y se le da nuevamente el grado de ajuste polinomial. y se crean los modelos en base a los datos que fueron guardados de manera aleatoria en las listas anteriormente mencionadas.

fbt1 = sp.poly1d(np.polyfit(xb[train], yb[train], 1))
fbt2 = sp.poly1d(np.polyfit(xb[train], yb[train], 2))

Se toman todos los **xb** y las **yb** y usando un polinomio de segundo grado, se debe encontrar una función tal que el error sea mínimo y después se debe construir la ecuación matemática y esta seria fbt2; esta función optimiza los valores.
Y se realiza lo mismo para los polinomios de grado tres hasta el centésimo.
fbt3 = sp.poly1d(np.polyfit(xb[train], yb[train], 3))
fbt10 = sp.poly1d(np.polyfit(xb[train], yb[train], 10))
fbt100 = sp.poly1d(np.polyfit(xb[train], yb[train], 100))

se realiza Tambien la evaluacion del modelo donde se evidencian los errores despues del punto de inflexion. Pero Tambien se realiza un blucle For donde se determinan los errores que se presentar en los modelos después de hacer el ajuste y analizarlos en la nueva lista de los valores aleatorios que se guardaron en train.
for f in [fbt1, fbt2, fbt3, fbt10, fbt100]:
se imprimen los errores en orden, analizándolos en cada modelo en un punto de la lista. que se guardó y de manera organizada en test.

print("Error d=%i: %f" % (f.order, error(f, xb[test], yb[test])))

Para el analisis que se realize anteriormente Tambien se crea nuevamente una grafica donde se pueden ver apreciados todos los ajustes polinomiales que se realizaron.

plot_models(
x, y, [fbt1, fbt2, fbt3, fbt10, fbt100],
os.path.join():
El método en Python une uno o más componentes de ruta de forma inteligente
os.path.join(CHART_DIR, "1400_01_08.png"),
mx=np.linspace(0 * 7 * 4, 8 * 7 * 4, 10000),
ymax=10000, xmin=0 * 7 * 4)

print (fbt2)
print (fbt2 - 100000)
se impreme en pantala la function de Segundo grado, pero Tambien se imprime restandole el valor de la prediccion que quiero realizar.
el polinomio de grado 2 (fbt2-100.000) y el valor que tomamos como base y lo dividimos por (7*4) para pasar el resultado en meses.
from scipy.optimize import fsolve print(fbt2)

proporciona funciones para minimizar (o maximizar) funciones objetivo, posiblemente sujetas a restricciones.
Incluye solucionadores de problemas no lineales (con soporte para algoritmos de optimización locales y globales), programación lineal, mínimos cuadrados restringidos y no lineales, búsqueda de raíces y ajuste de curvas.
de esta librería se importa la función fsolve: función que me permite hacer una estimación del resultado encuentra las raíces de una función.
Devuelve las raíces de las ecuaciones (no lineales) definidas por una estimación inicial dada.
alcanzado_max = fsolve(fbt2 - 100000, x0=2000) / (7 * 4)
print("\n100,000 Contagios esperados en el mes %f" %
alcanzado_max[0])
y por último se muestra el resultado de en que mes se espera el numero de contagios por el cual le preguntamos anteriormente al problema, y este nos devolverá en que mes se verán reflejados esos contagios si la curvatura de los datos planteados en la grafia sigue con la misma tendencia de aumentar.
print("\n100,000 número de contagios esperados en la mes %f" % alcanzado Max[0])

III. RESULTADO

Después de hacer el análisis del código anterior se debe mostrar cuales fueron los resultados obtenidos en el proceso.

Primeramente, se muestra los primeros 20 datos, leídos del archivo anteriormente mencionado “datosCovid.tsv” en el resultado obtenido se puede apreciar en una columna el número de días y en la otra se puede apreciar el numero de contagios obtenidos en ese día, se aclara que en esos datos que se proporcionaron son solamente el número de contagios, pero en el departamento de Risaralda.

```
[[nan nan]
 [ 1.  1.]
 [ 2.  2.]
 [ 3.  5.]
 [ 4.  6.]
 [ 5. 12.]
 [ 6. 13.]
 [ 7. 16.]
 [ 8. 18.]
 [ 9. 18.]
 [10. 18.]
 [11. 18.]
 [12. 28.]
 [13. 28.]
 [14. 34.]
 [15. 34.]
 [16. 34.]
 [17. 36.]
 [18. 36.]
 [19. 36.]]

(Nro de filas, Nro de columnas)
(190, 2)
```

También se puede apreciar que me muestra el número de filas que posee la matriz y el numero de columnas.
Después se verifica si dentro de la matriz existe algún numero o valor de tipo NAN y se puede evidenciar que existen una entrada incorrecta.

```
Número de entradas incorrectas: 1
```

Teniendo en cuenta y con base a los datos suministrados para el ajuste polinomial, generan principalmente una grafica inicial donde se aprecia todos los datos suministrados en el orden del mes en que fue tomado.

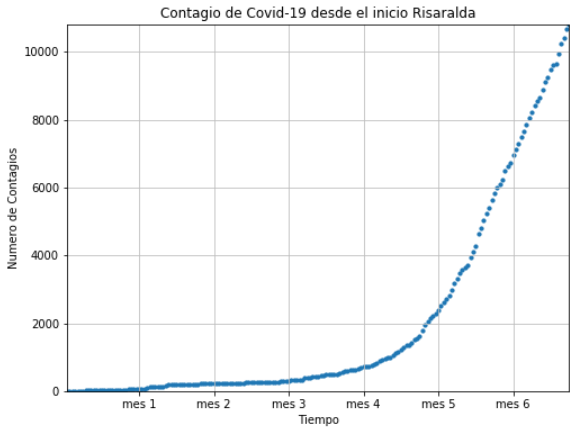

Figura 0

Los ajustes de grados polinomiales mediante la función def plot_models(x, y, models, fname, mx=None, ymax=None, xmin=None) genera como resultado ajuste de puntos mediante diversos tipos de líneas tales como recta, curva etc

Cuando se realiza el primer ajuste polinomial, se puede apreciar cuando se traza una recta en la gráfica, tratando de ajustarse a los puntos, que son los datos que se encuentran de manera agrupada en la gráfica y se puede apreciar en la siguiente ilustración.

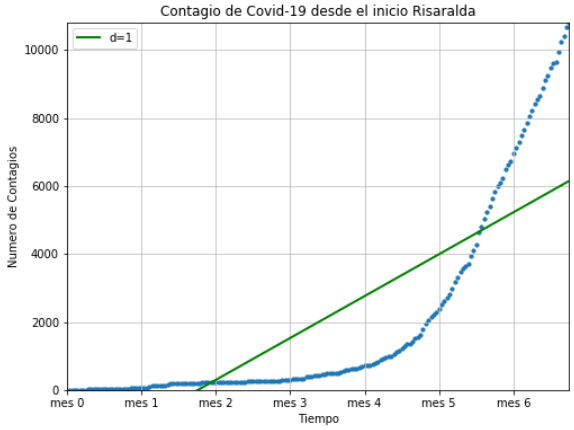

Figura 1

Se debe tener en cuenta que este no es el único ajuste de puntos, ya que se implementan dentro del programa diversos grados polinomiales tal como el grado dos que dibuja una gráfica:

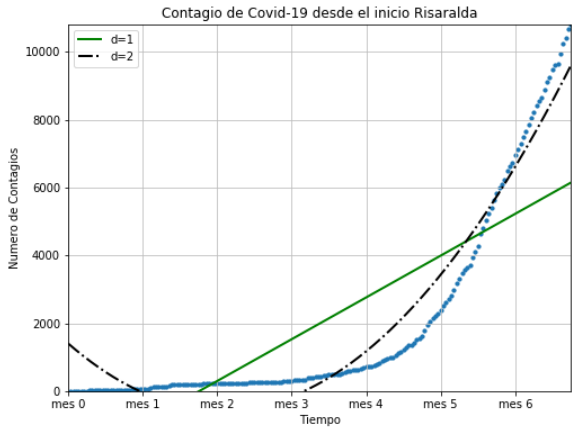

Figura 2

Como se aprecia en la figura anterior la segunda curva de color negro que es el ajuste polinomial de grado 2, se ve que tiende a ajustarse mas a la curvatura de los datos en la gráfica.

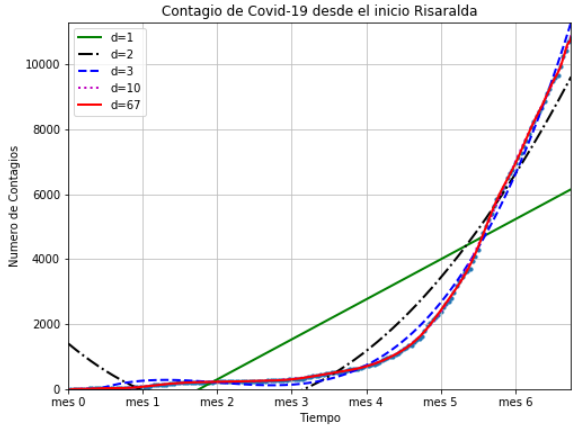

Figura 3

En la figura anterior se muestran tres ajustes polinomiales mas donde vemos que el ultimo ajuste polinomial se comporta de la misma manera que los datos de que se proporcionaron para el análisis.

Para las graficas anteriores de los modelos que se dieron también presentaron unos errores que se evidencian de la siguiente manera.

```
Parámetros del modelo fp1: [   43.9770
97   -2161.50146347]
Error del modelo fp1: [5.54867856e+08]
Parámetros del modelo fp2: [ 5.9026346
0e-01 -6.81729604e+01 1.40860870e+03]
Error del modelo fp2: [88135917.186699
96]
```

Mediante el método

```
plt.xticks(
    [w * 7 * 4 for w in range(20)],
    ['mes %i' % w for w in range(20)])); nos permite tener
```

como resultado las etiquetas y las marcas del grafico tales como son mes 1,mes 2, y mes 3 y los consecutivos, los cuales hacen referencia a los meses de marzo hasta el mes de septiembre estos meses indican la cantidad de personas que están contagiadas con el virus del coronavirus.

El método autoscale es el que permite obtener cuadriculas dentro de las gráficas para este caso tienen un color de 0.75 y se encarga de guardarlas.
Y que se menciona anteriormente en el método de la enseñanza.

```
plt.autoscale(tight=True)
    plt.ylim(ymin=0)
    if ymax:
        plt.ylim(ymax=ymax)
    if xmin:
        plt.xlim(xmin=xmin)
    plt.grid(True, linestyle='-', color='0.75')
    plt.savefig(fname)
```

En las gráficas presentadas anteriormente se evidencia el comportamiento de los datos, pero sin tener en cuenta el punto de inflexión, que es cuando los datos tienden a variar de forma que se evidencia un aumento de estos datos y se puede apreciar lo siguiente.

Podemos apreciar que hay dos líneas que se cruzan en la grafica indicando en donde se presenta el punto de inflexión de los datos.

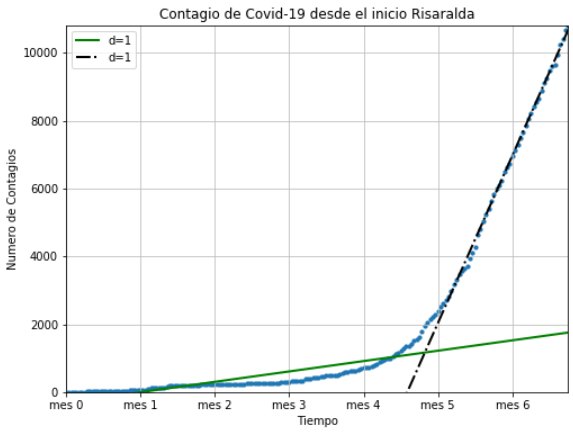

Figura 4

En la figura podemos apreciar que el punto de inflexión de los datos y a partir de este se realiza un conjunto de pruebas de análisis y determinar el error que se presenta en el conjunto de los datos suministrados.

- Conjunto de errores que resultaron de los datos

```
Error d=1: 554867856.206986
Error d=2: 88135917.186700
```

```
Error d=3: 7375531.219383
Error d=10: 314115.638545
Error d=67: 140574.168237
```

- conjunto de errores solamente después del punto de inflexión

```
Error d=1: 261552977.797269
Error d=2: 25188924.526943
Error d=3: 3681262.926644
Error d=10: 253979.095270
Error d=67: 104738.406669
```

Para todos los errores anteriores de la inflexión se suman y se determina uno solo y se presenta como un total de la siguiente manera:

```
Error de inflexión=12532690.443950
```

Después de haber verificado los diferentes tipos de errores se grafica los datos. Para hacer proyecciones de casos covid-19 a futuro.:

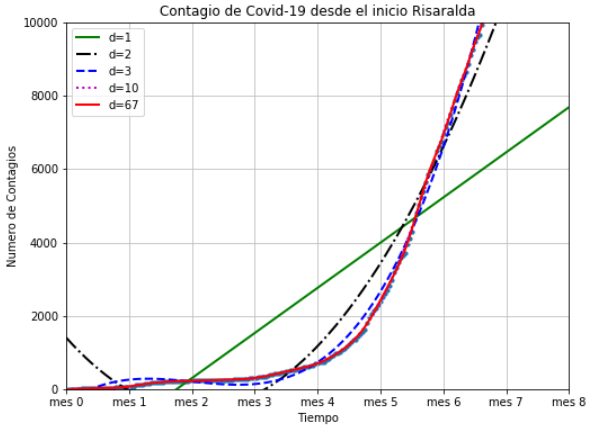

Figura 5

En este apartado es muy importante dado a que se entrenara el modelo. Se muestran en pantalla los errores, pero después de alcanzar el punto de inflexión.

```
Entrenamiento de datos únicamente
después del punto de inflexión
Errores después del punto de inflexión
Error d=1: 550361.039898
Error d=2: 441642.399168
Error d=3: 390863.351893
Error d=10: 100363.297161
Error d=67: 46041.277350
```

Los errores producidos después del punto de inflexión, en donde se aprecia que los errores disminuyeron, ya que solo se esta evaluando unos puntos de muestra.

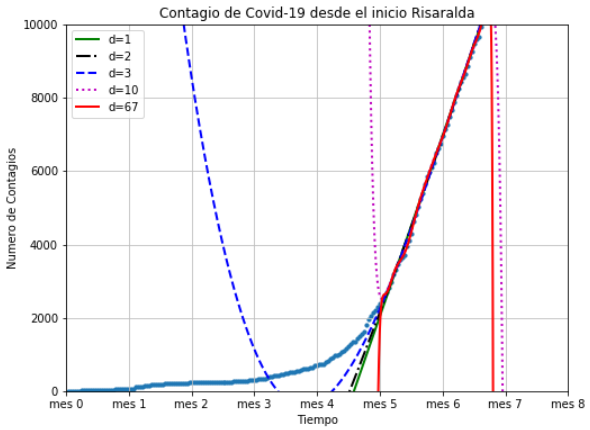

Figura 7

La grafica que se genera después de los puntos de inflexión se muestran en la parte superior.
Consiguiente a las gráficas mostraremos cuales son los modelos cuadráticos generados con el análisis de los datos.

Se muestra en pantalla a fbt2 que es la función original y el resultado de fbt2 menos -100000.

```
fbt2(x)=
          2
0.2842 x + 81.66 x - 1.481e+04
fbt2(x)-100,000=
          2
0.2842 x + 81.66 x - 1.148e+05
```

Cuando se está realizando el análisis de los datos en los modelos cuadráticos también se generan o se imprimen los errores que estos generan después del punto de inflexión junto con los modelos y se evidencia su resultado.

```
Prueba de error para después del punto
de inflexión
Error d=1: 115688.334318
Error d=2: 105393.800951
Error d=3: 83519.173779
Error d=10: 42558.771558
Error d=67: 38746.962645
```

Con estos grados polinomiales también se generan dos ecuaciones de grado 2, esto se realiza en el punto donde se va a realizar la predicción de en qué mes se esperan los 100 mil contagios de covid en el departamento de Risaralda.

```
          2
0.2842 x + 81.66 x - 1.481e+04
          2
0.2842 x + 81.66 x - 1.148e+05
```

Las dos funciones presentadas anteriormente son las dos funciones de grado 2 que se analizaban para la predicción de los contagios.

Después de esto se presenta la predicción que realizo el programa indicando en que mes se pueden presenciar el número de contagios preguntado.

```
100,000 contagios esperados en el mes
18.142290
```

Por ultimo se presenta la siguiente grafica donde se finaliza la predicción y como los ajustes polinomiales se comportan de una manera similar al de los datos presentados.

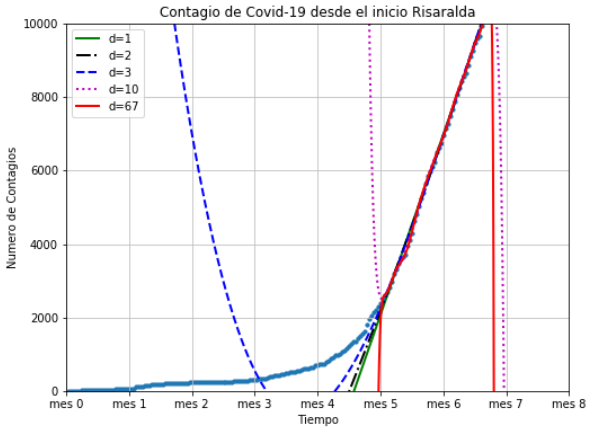

Figura 8

IV. CONCLUSIONES

- Según el análisis realizado se puede determinar que para el departamento de Risaralda el aumento de los contagios se ha presentado desde el cuarto mes en lo adelante y de seguir esta tendencia los contagios se podrían propagar muy rápido saturando los sistemas de salud

- La predicción generada por el programa sigue una tendencia de aumento que se ha verificado y comparados con los contagios que se suben en al base de datos de la página oficial del departamento de Risaralda, también haciendo lectura de los documentos subidos en la misma plataforma se evidencia que los contagios están en aumento y la saturación de la ocupación de las unidades UCI se encuentran en un 87%, confirmando un poco la tendencia de la predicción dada por el programa.

- Se puede apreciar también que el programa funciona de una manera adecuado y que aprende de buena forma dándole la matriz de los datos que se requieren para que el funcione analice y aprenda.

V. INFORMACION ACADEMICA

- **Ana Manuela Gamboa Piedrahita**

nació el 26 de julio de 1998 en Pereira, Risaralda, madre Consuelo Piedrahita y padre Milton Gamboa.
estudio de 2003-2015 en la institución educativa el Dorado graduada el 06 de diciembre de 2015 con el mejor ICFES de la jornada de la mañana.
Estudio de 2014-2015 un técnico en desarrollo de software en el Sena durante sus últimos años de escolares.
Estudiante de ingeniería de sistemas y computación de la universidad tecnológica de Pereira (UTP), iniciando el año 2017, ganadora de una beca por parte del programa becas para pepas, creado en el 2017, por la alcaldía de Pereira.

- **Orfilia castillo Maturana,** nació el 24 julio en Tado Colombia 1998, graduada del bachillerato en el 2015 de la institución educativa el dorado con mención de honor por ser una de las estudiantes más disciplinadas, en el 2015 obtuvo también el título de tecnóloga en programación del software de la institución de educación SENA de la ciudad de Pereira.

Becada por la Alcaldía de Pereira, y actualmente estudiante de ingeniería en sistemas y computación en la universidad tecnológica de Pereira y cursando el octavo semestre, estudiante también de contabilidad y finanzas de manera virtual en la institución a Distancia SENA, y trabaja en la floristería ml detalles de la ciudad de Pereira ubicada en el sector del poblado etapa 1.