Regresión lineal con Python aplicado a Casos de COVID-19 en Colombia 06 marzo a 14 septiembre 2020

Linear regression with Python applied to COVID-19 cases in Colombia March 06 to September 14, 2020

Autor: Julian Giraldo C
J. David Álvarez

IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: julian.giraldo2@utp.edu.co
j.alvarez2@utp.edu.co

***Resumen*— Este documento presenta una forma de aplicar Machine Learning mediante el lenguaje de programación Python, El proyecto está enfocado al procesamiento de una serie de datos del COVID-19, el documento de los datos va desde Marzo 06 hasta el mes de Septiembre 14 analizando los datos mediante un pre procesamiento y de esta forma se utiliza eficazmente la librería Numpy y sus funciones con un enfoque directo al manejo de matrices y vectores bidimensionales. Primero se hace una introducción del manejo de las mismas, también se aplica un método de predicción para saber ¿Qué pasará al finalizar las dos siguientes semanas, en cuanto al volumen de contagiados?. Todo esto mediante una serie de datos que se obtendrán desde una página oficial de información con el fin de aplicar la librería numpy, matplot, scipy y sklearn.**

***Palabras clave*— Python, funciones, matrices, machine learning, programación, procesamiento de datos, archivos, predicciones, regresión lineal.**

***Abstract*— This document presents a way of applying Machine Learning through the Python programming language, The project is focused on the processing of a series of COVID-19 data, the data document goes from March 06 to September 14 analyzing the data Through pre-processing and in this way, the Numpy library and its functions are used effectively with a direct approach to the management of two-dimensional matrices and vectors. First, an introduction to their management is made, a prediction method is also applied to know what will happen at the end of the next two weeks, in terms of the volume of infected. All this through a series of data that will be obtained from an official information page in order to apply the numpy, matplot, scipy and sklearn library.**

***Key Words*— python, functions, arrays, machine learning, programming, data processing, files, predictions, linear regression.**

I. INTRODUCCIÓN

Los coronavirus se descubrieron en los años 60 siendo importantes patógenos humanos y animales, provocando distintas enfermedades que pueden ir desde un resfriado hasta una neumonía. Hasta diciembre del 2019, se habían identificado seis tipos de coronavirus que pudieran generar enfermedad en humanos, entre ellos los causantes de los dos brotes epidémicos anteriores: el SARS coronavirus que apareció por primera en el año 2002 y el MERS-CoV, que se identificó por primera vez en el año 2012 en el medio oriente. A finales de diciembre del 2019, se identificó un nuevo coronavirus como el agente causal de un grupo de casos de neumonías en Wuhan, capital de la provincia de Hubei en China, denominándolo la Organización Mundial de la Salud (OMS) en febrero de 2020, coronavirus 2 del síndrome respiratorio agudo severo (SARS-CoV-2) y a la enfermedad que origina COVID-19, que significa enfermedad por coronavirus 2019. Desde Wuhan se extendió rápidamente, dando como resultado al inicio una epidemia en toda China, seguida de un número creciente de casos en todo el mundo, generado la pandemia y emergencia sanitaria actual.

Al ser una patología reciente aún se desconoce mucho de su epidemiología, transmisión, tratamiento, etc. Se están realizando continuos estudios para profundizar en su conocimiento.

Se sabe que este virus Covid-19 estará presente por mucho más tiempo así que queda perfecto para hacer la investigación acerca de predicción mediante regresión lineal, la pertinente investigación fue llevada a cabo con las cifras oficiales de casos confirmados en el país de Colombia, con un registro diario.
Machine Learning es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. Aprender en este contexto quiere decir identificar patrones complejos en millones de datos. La máquina que realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros. Automáticamente, también en este contexto, implica que estos sistemas se mejoran de forma autónoma con el tiempo, sin intervención humana.

Numpy es actualmente la mejor manera de trabajar con Arrays, Vectores y Matrices, en Python. Es una de las librerías más eficientes en CPU para hacer operaciones con matrices y convertir Python en algo más parecido a Matlab y acelerar Python al máximo. Las predicciones juegan un papel importante en el mundo de las estadísticas, esto principalmente para tomar decisiones anticipadas que generen cambios positivos, dichas predicciones se pueden lograr gracias a los diferentes tipos de algoritmos que existen en Python como la librería sklearn.

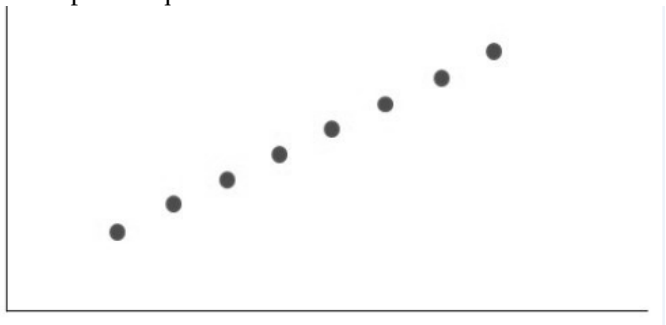
II. OBJETIVOS

- Repasar conceptos básicos de Numpy.
- Resolver un proyecto sobre COVID-19 utilizando predicciones mediante regresión lineal.
- Manejo de datos.
- Graficación de datos.
- Manejo de puntos de inflexión

III. METODOLOGIA

Para realizar una predicción correctamente se tienen que aplicar distintos métodos de regresión lineal, la regresión lineal consiste en trazar una recta en una serie de puntos en un espacio determinado, en este caso nuestro espacio es un plano cartesiano de 2 dimensiones.

Como se observa en la fig 1. Tenemos 8 puntos que en este caso están unos seguidos de otro formando una recta así que la regresión lineal se aplica de manera muy sencilla, simplemente trazando dicha recta por cada uno de los puntos que se encuentran allí.



Fig, 1 La recta mostrada se adapta a los puntos

Gracias a esta recta de la Fig. 1 podemos saber que sucederá a futuro que se denomina: XF, y el valor en y es lo que se muestra en la Fig. 2

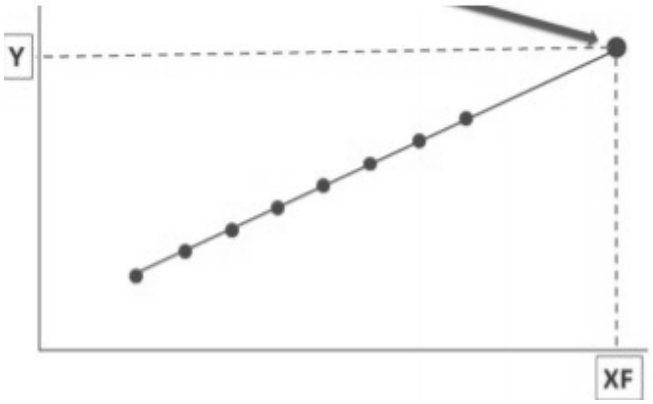


Fig. 2 Punto que determina la predicción

La flecha que señala ese punto rojo indica la predicción en un plazo cercano, es fácil conocer las predicciones gracias a la tendencia de dicha recta, pero ahora bien. ¿Cómo sería una predicción para una función parabólica o una gráfica de dispersión?, Esta pregunta será tomada en cuenta en las próximas explicaciones.
Ahora supongamos que la recta se hizo muy aparte de los puntos, entonces quedaría de esta forma Fig. 3

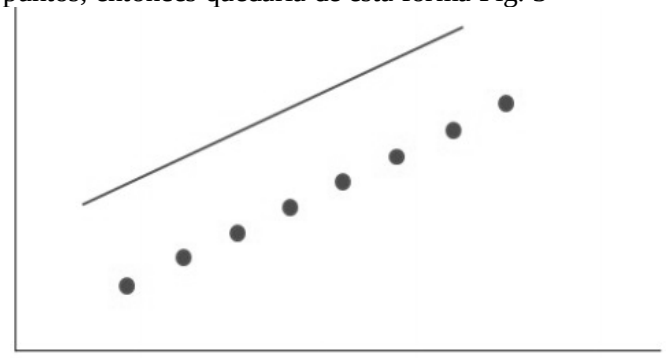


Fig. 3 Recta con error

La recta al ser desplazada ya no se ajusta a los puntos, la razón es que existe una diferencia entre los puntos de la recta que trazamos y los puntos que hay en la Fig 3. De esta forma se genera un "error" que se verá en la Fig 4.

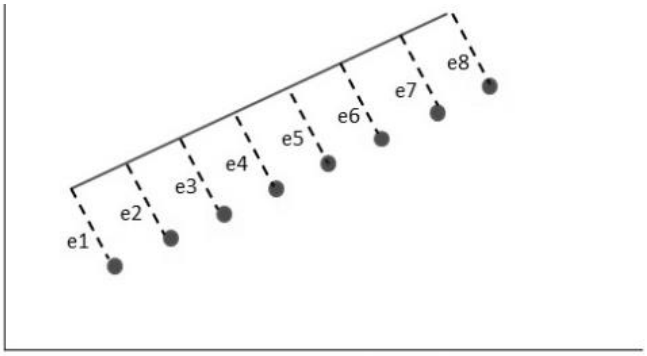


Fig. 4 Recta con cada error en el punto

Las líneas que conectan a cada punto con la recta representan el ERROR generándose así en cada uno de los puntos de ajuste, Los errores son todos los valores "en". Mientras mayor sea la suma, mayor es el error.

```
Error= e1+e2+e3+e4+e5+e6+e7+e8
```

Ahora podemos retomar con la pregunta, ¿Cómo sería una predicción para una función parabólica o una gráfica de dispersión?, la recta que se acomodaría mejor a esto es la que genere un menor error posible, la gráfica de dispersión está en la Fig. 5.

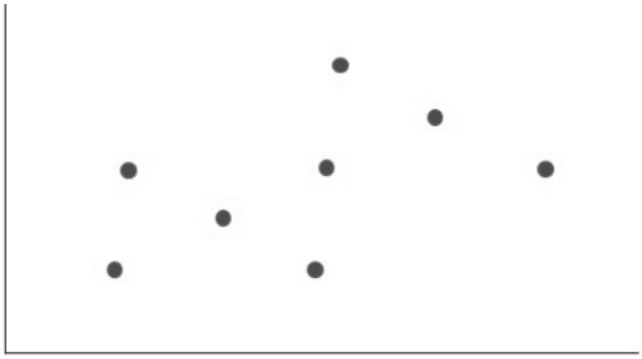


Fig. 5 Gráfico de dispersión

Si en esta gráfica se traza una recta como se ha hecho en las figura, se podría pensar que existe una recta que puede abarcar la mayor parte de los puntos de una u otra manera. Pero para llegar a esto se necesita cambiar el orden polinomial ya que una recta no se puede acoplar a cambios mínimos como una curva.

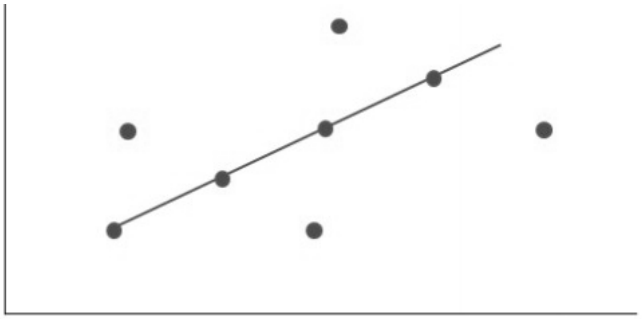


Fig. 6 Trazo de una recta en gráfico de dispersión

Se puede pensar que con esa recta trazada en la Fig 6. Se pueden realizar predicciones como se ha hecho anteriormente pero este modelo es inexacto de cierto modo.

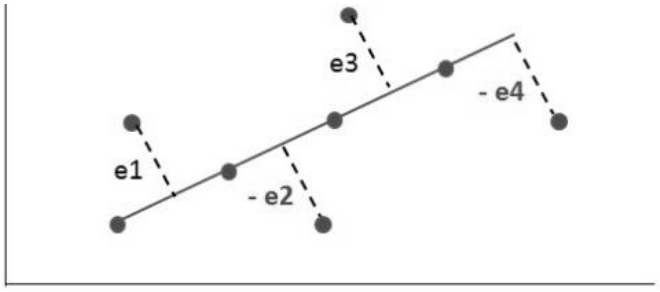


Fig. 7 Hallando el error en la recta de gráfico dispersión

Al trazar cada recta punteada para saber el error en esos puntos que están presentes en la Fig. 7 podemos determinar de cual magnitud es el error, se puede observar que hay errores positivos como negativos así que esto puede generar un reto a la hora de sumar el error total ya que un número positivo se resta con el negativo y el resultado del error nos daría 0 y esto es un resultado alejado de la realidad.

```
Error= e1-e2+e3-e4 = 0
```

Una solución a este problema es elevar todos los valores de error al 2 "$(e_n)^2$" para así eliminar los signos negativos pero hay que tener muy en cuenta que se hizo ese cambio en cada valor.

Una de las alternativas para establecer la recta más adecuada que pueda conectarse con la mayor parte de los puntos es utilizando una ecuación cuadrática o mejor aún un polinomio de grado n.

Utilizando un polinomio reduce drásticamente el error y así mejorar la respuesta del sistema para futuras predicciones acertadas como se observa en la Fig. 8

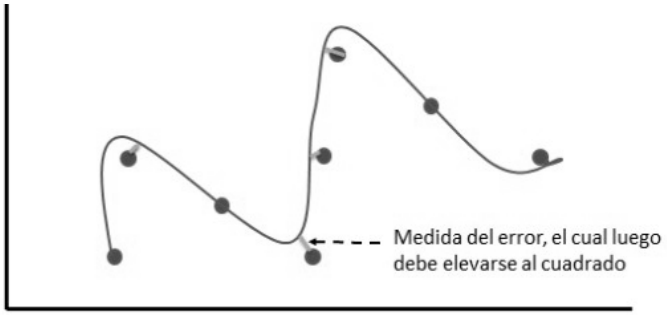


Fig. 8 Ajustamiento de curva

En la Fig. 8 se demuestra cómo puede quedar la línea recta con un orden polinomial mayor que 1 y de esta forma reducir el error.

Ahora que se entiende mejor el término de error mediante regresión lineal podremos aplicar esto a código de programación que en este caso será Python, primero comenzamos importando las librerías necesarias para el procesamiento de los datos.

```
Import os
```

Primero que nada importamos la librería llamada "os" que es el sistema operativo y esto ¿con qué fin se hace? Con el fin de crear unos directorios donde serán almacenadas las gráficas que realizaremos más adelante y también el directorio de los datos. Esto se hace gracias a la instrucción

```
from utils import DATA_DIR, CHART_DIR
```

Luego de importar las librerías de directorio del sistema podemos importar las librerías restantes, librería sklearn modelo lineal para realizar predicciones, numpy para trabajar con arreglos y matrices, scipy y matplotlib.

Se importa el archivo con el que vamos a trabajar respectivamente, dicho archivo contiene 2 columnas donde la primer columna son los días y la segunda columna son los casos confirmados de COVID-19, se llevó un registro de 193 días donde los casos confirmados estuvieron en un rango moderado sin sobrepasar los 13000 casos. Gracias a esta instrucción se puede obtener el archivo en el directorio DATA especificado anteriormente.

```
data=np.genfromtxt(os.path.join(DATA_D
IR,"COVID-19cases.tsv"),
delimiter="\t")
```

Después de obtener el archivo en la variable data, ejecutamos la instrucción para colocar todos los datos en un float64, esto con el fin de reducir los números de muchas cifras ya que, como estamos manejando datos reales de covid puede superar hasta los 10000.

```
data= np.array(data, dtype=np.float64)
print(data[:10])
print(data.shape)
```

Se establece cada valor de data en una notación científica utilizando el :10. Ahora se definen los colores que tendrán los puntos y las rectas de regresión lineal en las gráficas.

```
colors = ['g', 'k', 'b', 'm', 'r']
```

La letra 'g' indica el color Green, 'k' black, 'b' blue, 'm' magenta y 'r' red, se especifican varios colores ya que vamos a tener diferentes tipos de recta y de esta forma se pueden diferenciar más fácil unas de otras. Así mismo se definen los tipos de líneas que tendrán las rectas y la maya de la gráfica.

```
linestyles=['-', '-.', '--', ':', '-']
```

Luego de establecer los colores y el tipo de línea, dividimos los datos que tenemos de COVID en 2 arreglos para facilitar su manejo a la hora de procesar los valores.

```
x = data[:, 0]
y = data[:, 1]
```

El archivo en este caso tiene que tener únicamente 2 columnas, si tiene más de 2 columnas puede generar un error ya que en este caso estamos extrayendo únicamente los datos de la columna 0 que equivale a la 1 y la columna 1 que equivale a la 2.

Se crean dos nuevas variables xp y yp para realizar las respectivas predicciones mediante la librería sklearn, cabe resaltar que estas predicciones no son tan efectivas ya que realizan un cálculo numérico dependiendo de los datos que hay internamente así que no predice perfectamente en un futuro muy lejano. Se crea una instancia del modelo de regresión lineal.

```
model=LinearRegression()
```

Luego de crear esta instancia lo que se procede a hacer es acomodar los valores que tenemos en las variables Xp y yp para así entrenar a la variable para que pueda realizar predicciones.

```
model.fit(Xp,yp)
```

Se crean unas variables para establecer los días que queremos predecir ya que los datos que tenemos están registrados por días. También necesitamos saber los días totales que tenemos registrados en nuestros datos y esto se puede lograr gracias a la función len(). Otro valor que nos interesa conocer es el total de contagiados, con la instrucción `np.sum()` podremos saber el total de elementos en un arreglo.
Ahora que tenemos los datos que nos importa, procedemos a hacer la predicción con los días que queremos predecir.

Se crea un ciclo mediante la función While donde allí se pondrán los días que no están registrados en la base de datos para así irlos sumando con el total de contagiados.

```
test_covid=np.array([[diaspredic]])
```

Se crea un arreglo llamado test_covid y allí se aloja la información de una matriz que lleva lo días que queremos predecir, los diaspredic varían desde el último día de registro de contagios hasta el límite que se indicó así que este valor estará cambiando en el ciclo while.

```
contagiados_predict=model.predict(test
_covid)[0]
```

Ahora vamos a la predicción, con la instrucción model.predict podemos predecir el valor más cercano a esos días que especificamos y de esta manera se guardan en la variable contagiados_predict.

Normalmente se hace un análisis a los datos extraídos del archivo con la instrucción.

```
np.isnan(y)
```

En este caso no hubo ningún valor nan en los datos que tenemos registrados ya que nuestra fuente fue una página de estadísticas así que dichos datos ya pasaron por un proceso de filtro.
El resultado que nos arroja la predicción es un valor que va en constante aumento, no importa si se estiman 1000 días, cada vez aumenta más lo que parece un poco ilógico.

```
El total de casos acumulados en 300 días
son: 2143348.0479808766
```

El resultado de la predicción para 300 días de casos acumulados supera los 2 millones de casos.

Luego de establecer la predicción procedemos a crear una clase donde graficaremos todos los datos en modo de dispersión y también aplicaremos regresión lineal para ver como actúa cada recta dependiendo del grado del polinomio.

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

plot_models recibe el arreglo x, el arreglo y, el tipo de modelo que se quiere realizar (gráfica), el nombre de la función, la pendiente, el valor máximo en y, y el valor mínimo en x.

```
plt.figure(num=None, figsize=(8, 6))
```

Se crea una nueva figura donde recibe la variable num que es el identificador, y el figsize que indica el tamaño de la figura.

```
plt.clf()
```

Con la función plt.clf() se borra el espacio de la figura para así dejar campo libre.

```
plt.scatter(x, y, s=10)
```

Se crea el gráfico con el arreglo x y el arreglo y, con un tamaño de 10 en cada uno de los puntos, luego se establecen los títulos en cada uno de los ejes, y el título del encabezado.

Ahora procedemos a especificar las etiquetas para cada uno de los valores en la gráfica.

```
plt.xticks(
        [w * 7 * 1 for w in range(28)],
        ['%i' % w for w in range(28)])
```

La variable w toma los valores en el rango de 28 lo que significa 28 semanas y a su vez se va multiplicando con el número 7, esto significa que el gráfico será etiquetado por semanas durante 28 iteraciones.

Sabemos que el modelo de cada gráfico cambiará entonces colocamos una condición if para saber si se especificó el modelo o no.

```
if mx is None:
       mx = np.linspace(0, x[-1], 1000)
```

Pregunta que si la pendiente no existe entonces que realice un espacio de las líneas con los valores de 0, el arreglo x en la posición -1 y 1000.
Esas instrucciones se realizaron cuando el modelo no existe pero entonces, ¿Cómo sería la instrucción para un modelo especificado?

```
for model, style, color in zip(models,
linestyles, colors):
```

Este ciclo for utiliza la variable del modelo, el estilo de línea y el color, para interactuarlo con las librerías internas zip en ingles.

```
plt.plot(mx,model(mx),linestyle=style,
linewidth=2, c=color)
```

La instrucción plt.plot se utiliza para el modelado de la gráfica que estamos elaborando.

```
plt.autoscale(tight=True)
```

Con esta instrucción se hace un escalamiento de los datos para cada uno de los valores en la gráfica. Con la instrucción grid mostramos las líneas de la cuadricula. Si el valor está en true entonces esas líneas se van a proporcionar.

```
plt.grid(True,linestyle='-',
color='0.75')
```

Ahora que ya establecimos los valores para graficar cada uno de los datos de los casos acumulados de COVID-19 en Colombia aplicamos la instrucción para guardar la figura en el directorio que especificamos al principio del código.

```
plt.savefig(fname)
```

Recibe como argumento el nombre de la gráfica. Vamos a darle un primer vistazo a la graficación de los datos, sabemos que los datos están por días que en total son 193 pero en este caso para reducir la cantidad de variables x reducimos el formato a semanas que en total son 27 semanas.

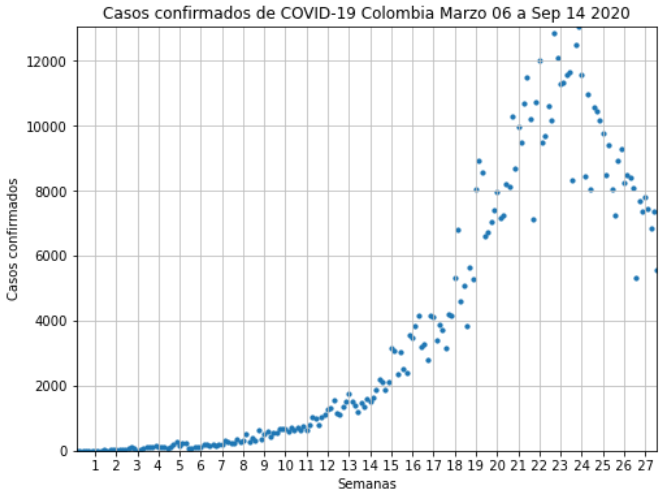


Fig. 9 primer vistazo a los datos COVID-19 Colombia

El gráfico indica un gran aumento a partir de la semana 13, 14 y llega a un tope en la semana 24, con la cantidad de datos que tenemos nos limitamos a observar solamente 27 semanas pero gracias a los puntos de inflexión y sus diversas aplicaciones podremos predecir qué sucederá en semanas futuras.
Ahora que verificamos que el gráfico muestra los datos de manera correcta, podemos aplicar regresión lineal de un orden polinomial 1 con la siguiente instrucción.

```
fp1,res1,rank1,sv1,rcond1=np.polyfit(x
, y, 1, full=True)
```

np.polyfit es una función para el ajuste polinomial de mínimos cuadrados con las entradas x, y, el valor 1 indica el grado.

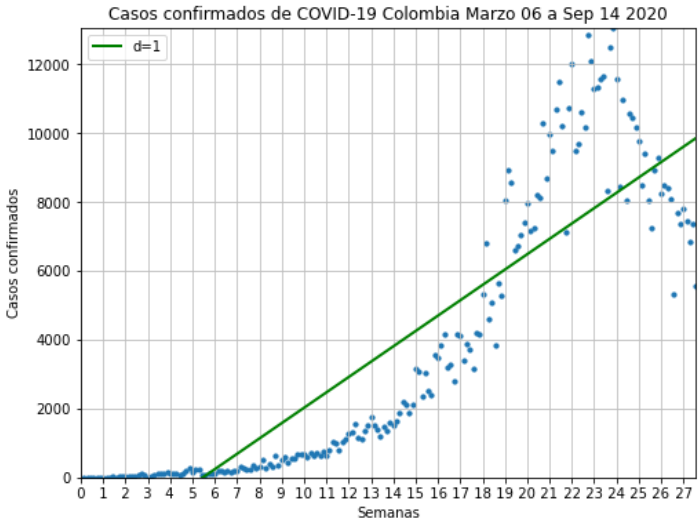


Fig. 10 Regresión lineal con recta de orden 1

Como se explicó en la Fig. 4, vamos a calcular el error de esta función fp1 para saber qué tan alejados están los puntos de la recta que acabamos de trazar. Dicho error se almacena en la variable res1. El resultado es 6.71773991e+08, ya se esperaba este resultado porque con solo observar la recta, se nota que está demasiado alejada de la mayor parte de los puntos.
Ahora, realicemos una gráfica de orden de polinomio 2, y calculemos el error.

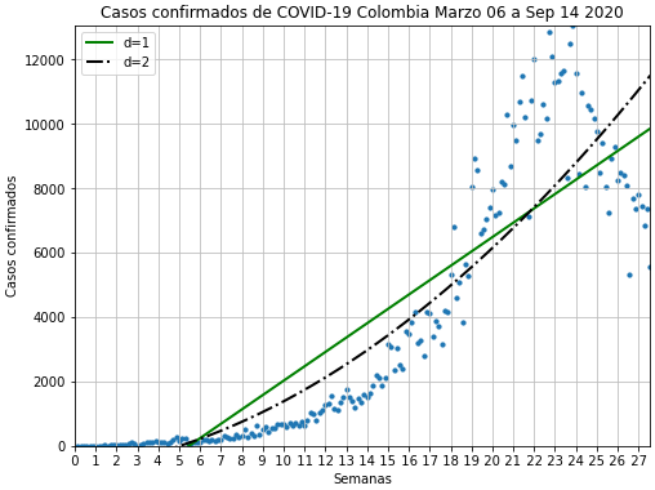


Fig. 11 Regresión lineal con recta de orden 2

Se nota un ligero cambio en la recta numero dos ya que esta tiene un grado mayor pero no lo suficiente para abarcar la totalidad de los puntos en la gráfica así que desde ahora podremos suponer que el error seguirá siendo un número muy grande.
El error en esta función de orden 2 es, 5.63769899e+08.
Tan solo se redujo en una unidad a comparación de la recta de color verde, esto es algo para nada notorio. En este orden de ideas la mejor opción sería una función de orden 100, eso lo probaremos enseguida.

```
f3 = sp.poly1d (np.polyfit(x, y, 3))
f10 = sp.poly1d (np.polyfit(x, y, 10))
f100 = sp.poly1d (np.polyfit(x, y, 100))
```

Con estas instrucciones graficamos unas funciones que van desde orden 3 hasta orden 100 y así podremos observar cual se acomoda más a la serie de puntos que tenemos en la gráfica. Desde ahora podremos suponer que la de orden 100 es la mejor función pero lamentablemente puede generar problemas a la hora de realizar las predicciones que deseemos hacer.

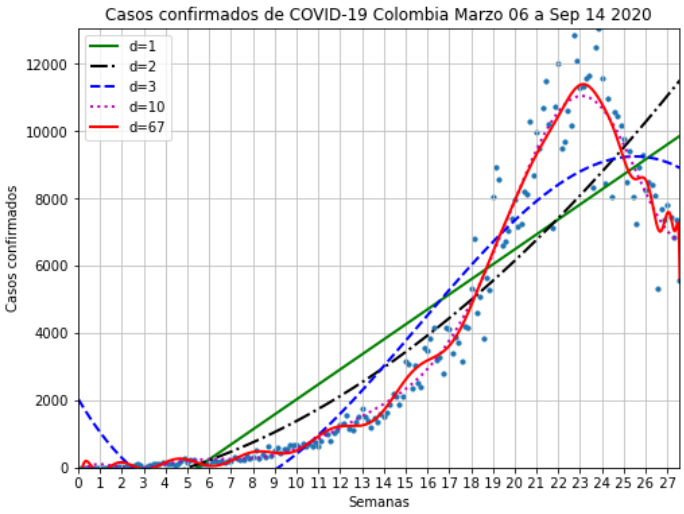


Fig. 12 Funciones de diferente orden

Se nota una gran diferencia entre cada función, es claro que cada vez que se aumenta el orden del polinomio se ve mejor su forma en que se acomoda a cada punto pero como se mencionó anteriormente esto puede ser una gran desventaja ya que una función de orden 100 tiene demasiados puntos de inflexión y esto afecta a la hora de querer predecir mediante regresión lineal, así que lo mejor es trabajar con la función de orden 2 para hacer un fácil manejo y así determinar un punto de inflexión fijo.

Hallando en punto de inflexión.

Un punto de inflexión, en una función matemática, es un punto donde los valores de una función continua en x pasan de un tipo de concavidad a otra. La curva "atraviesa" la tangente. Matemáticamente, la segunda derivada de la función f en el punto de inflexión es cero, no existe.

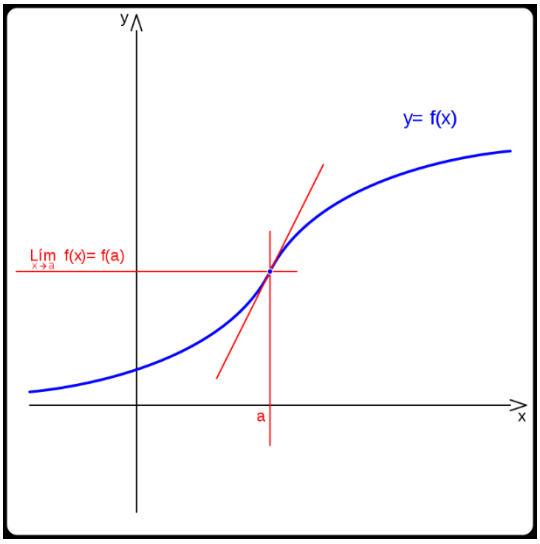


Fig. 13 Punto de inflexión

```
inflexion = 13 * 7 * 1
```

Con esta instrucción calculamos el punto de inflexión cada día

```
xa = x[:int(inflexion)]

ya = y[:int(inflexion)]

xb = x[int(inflexion):]

yb = y[int(inflexion):]
```

xa = Representa la primera mitad de los datos (1-91)

ya = Valores escalados que representan la segunda columna, y representan la primera mitad de los datos

xb = Representa la segunda mitad de los datos

yb = Valores escalados que representan la segunda columna, y representa la segunda mitad de los datos

```
fa = sp.poly1d(np.polyfit(xa, ya, 1))

fb = sp.poly1d(np.polyfit(xb, yb, 1))
```

Se grafican dos líneas rectas, con la instrucción np.polyfit para acomodar los datos haciendo uso de la librería numpy, esta recibe como argumento “xa” y “ya” y se indica el grado del polinomio “1”.

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

Se grafica el modelo basado en el punto de inflexión con uso de la clase plot_models que recibe como parámetros “x”,”y”,”fa” y “fb”, y guardamos la grafica con el nombre "1400_01_05.png"

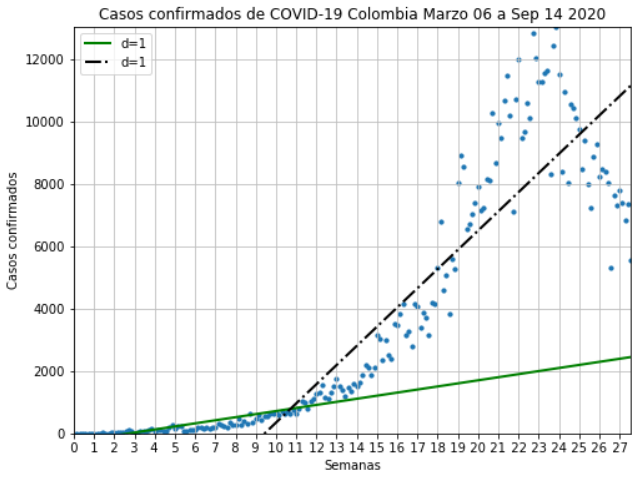


Fig. 14 grafica con punto de inflexión

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

Se define un método para determinar los errores de la función.

```
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)))

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)))

print("Error de inflexión=%f" % (error(fa,
xa, ya) + error(fb, xb, yb)))
```

Se imprimen los errores para el conjunto completo de datos, para los errores después del punto de inflexión y el error de inflexión, y también el error antes del punto de inflexión.

```
plot_models(
      x, y, [f1, f2, f3, f10, f100],
      os.path.join(CHART_DIR,
      "1400_01_06.png"),
      mx=np.linspace(0 * 7 * 1, 27 * 7 * 1,
      100),
      ymax=12000, xmin=0 * 7 * 1)
```

Se grafica utilizando el método plot_models enviado como argumento las variables “x”, “y” y las funciones f1, f2, f3, f10, f100.
ymax = máximo de datos que se van a graficar
xmin = mínimo de datos que se van a graficar

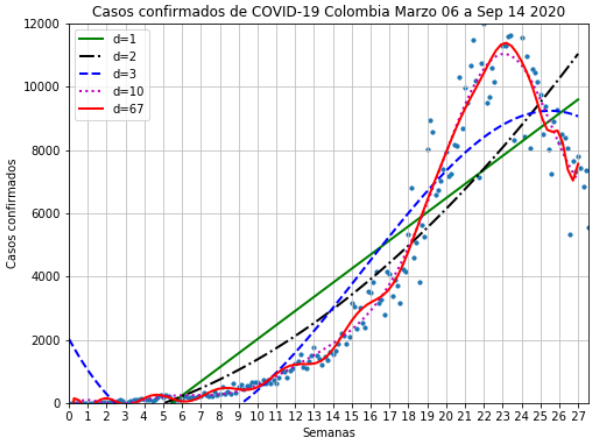


Fig. 15 grafica con diferentes funciones

```
fb2 = sp.poly1d(np.polyfit(xb, yb, 2))
```

Se entrenan los datos con los valores en el punto de inflexión haciendo uso de la instrucción np.polyfit, se indica el grado del polinomio, en este caso va desde 2 hasta 100.

```
for f in [fb1, fb2, fb3, fb10, fb100]:
    print("Error  d=%i:  %f"  %  (f.order,
    error(f, xb, yb)))
```

Se calcula el error con las funciones entrenadas, y las variables determinadas en 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 * 1, 27 * 7 *
      1, 100),
      ymax=12000, xmin=0 * 7 * 1)
```

Se crea una gráfica con las funciones entrenadas.

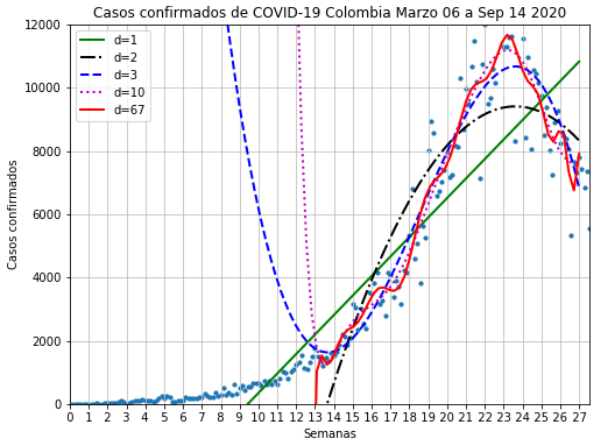


Fig. 16 grafica con funciones entrenadas

Se separa los datos de entrenamiento de prueba

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

Se toman los datos entrenados de la primera columna y se ordenan de manera aleatoria

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

Se crea la variable test que llama el método sorted para desorganizar los datos utilizando split_idx.

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

La variable train hace los mismo que la variable test pero este hace uso de los primeros datos de split_idx.

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

Se crea una function de nombre “fbt2” para entrenar mejor los datos determinados en el punto de inflexion, aquí se hace uso de la variable train y indicamos el grado del polinomio 2, de esta forma hacemos lo mismo con el resto de funciones.

```
for f in [fbt1, fbt2, fbt3, fbt10, fbt100]:
  print("Error  d=%i:  %f"  %  (f.order,
  error(f, xb[test],  yb[test])))
```

Calcula el error con las nuevas funciones super entrenadas después del punto de inflexión.

```
plot_models(
    x, y, [fbt1, fbt2, fbt3, fbt10,
    fbt100],
    os.path.join(CHART_DIR,
    "1400_01_08.png"),
    mx=np.linspace(0 * 7 * 1, 27 * 7 * 1,
    100),
    ymax=12000, xmin=0 * 7 * 1)
```

Se crea una gráfica con las nuevas funciones super entrenadas.

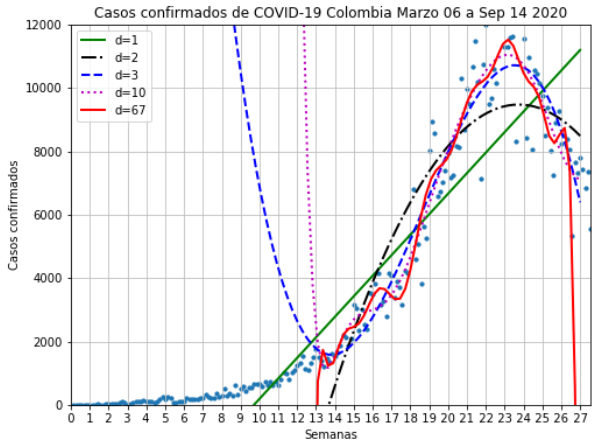


Fig. 18 grafica con funciones super entrenadas

```
from scipy.optimize import fsolve
```

Se importa la librería scipy.optimize

```
print(fbt2)
```

Se muestra a función super entrenada de grado 2

```
print(fbt2 - 100000)
```

Se muestra la función “fbt2” con la resta de 100000 para saber el momento en el que este valor de 100000 se encontrara en la posición “0” del eje Y, y poder saber que valor obtiene el eje X.

```
alcanzado_max = fsolve(fbt2 - 100000, x0=0)
/ (7 * 1)

print("100.000 casos%f" %alcanzado_max[0])
```

Se crea la variable alcanzado_max, que utiliza la función “fsolve” para resolver una ecuación matemática se divide entre 7 para obtener el valor en términos de semanas.

REFERENCIAS

Referencias en la Web:

[1] https://cleverdata.io/que-es-machine-learning-big-data/
[2] https://medium.com/@jcrispis56/machine-learning-parte-2-numpy-fe068abe5703
[3] https://techlandia.com/archivo-tsv-info_292135/
[4] https://www.arcgis.com/apps/MapSeries/index.html?appid=c8fee09df07f49f1aaf4f0f086bf7d8a
[5] https://ingenioempresa.com/regresion-lineal/
[6] https://es.wikipedia.org/wiki/Punto_de_inflexi%C3%B3n