Proyecto 1

Inteligencia Artificial

1st Jeffrey Orihuela
Computer Science
UTEC
Lima, Perú
jeffrey.orihuela@utec.edu.pe

2nd Juan Vargas
Computer Science
UTEC
Lima, Perú
juan.vargas@utec.edu.pe

***Index Terms*—linear regression, machine learning, multivariable**

I. Introducción

La regresión multivariada es aplicable en el momento en el que nuestro dataset es de más de 1 dimensión. En este informe se implementará este algoritmo con ayuda del lenguaje Python para poder predecir el área del terreno que podría incendiarse. Para este modelo nuestros usaremos distintas características como la ubicación, la fecha, temperatura, humedad, etc.

II. Explicación

El conjunto de datos describe una información relacionada a incendios forestales en la región norteste de Portugal, podemos ver estos datos en la figura 1. Las 2 primeras columnas describen las cordenadas espaciales en el parque Montesinho. Las dos siguientes columnas son meses y días, estos datos para nuestro propósito tuvimos que mapearlo a números. Las columnas 5,6,7,8 y 10 tienen intervalos altos los cuales deben ser normalizados para evitar problemas de vanishing gradient. La columna 8 representa la temperatura en grados Celsius, columna 11 representa el viento, columna 12 representa la lluvia. Por último, la columna 13 es el resultado de nuestra predicción y describe las areas quemadas en el parque Montesinho, esta columna tiene un problema de skewness por lo cuál debemos transformar la data mediante una función logaritmo natural. Una vez realizado la limpieza de datos podemos proceder a dividir los datos de acuerdo a las especificaciones del proyecto. Para este fin hemos aprovechado la función $train_test_split$ de la librería sklearn. Luego de separar el dataset en 70% para entrenamiento, 20% para validación y 10% para testing. Empezamos a trabajar nuestro entrenamiento usando el método gradiente descendiente. Nuestra forma de trabajar en este proyecto fue usando matrices para evitar el uso de loops en cada array, de esta forma podemos hacer uso del producto dot de la librería numpy. Mientras resolvemos nuestro entrenamiento vamos calculando el costo de error o función pérdida. Finalmente, graficamos el valor de la función costo por cada época realizada en el entrenamiento.

III. Experimentos

Para nuestra experimentación primero dividimos el dataset en 3 partes para las siguientes operaciones: entrenamiento (70%), validación (20%) y testeo (10%). Para esto usamos la función **train test split** de la librería *sklearn*.

Fig. 1. Dataset de los incendios forestales

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	3	5	0.870968	0.086492	0.101325	0.090909	8.2	0.423529	6.7	0.0	0.000000
1	7	4	10	2	0.927742	0.118194	0.775419	0.119430	18.0	0.211765	0.9	0.0	0.000000
2	7	4	10	6	0.927742	0.146795	0.796294	0.119430	14.6	0.211765	1.3	0.0	0.000000
3	8	6	3	5	0.941935	0.110958	0.081623	0.160428	8.3	0.964706	4.0	0.2	0.000000
4	8	6	3	7	0.910968	0.172984	0.110590	0.171123	11.4	0.988235	1.8	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
512	4	3	8	7	0.811613	0.191592	0.771315	0.033868	27.8	0.200000	2.7	0.0	2.006871
513	2	4	8	7	0.811613	0.191592	0.771315	0.033868	21.9	0.658824	5.8	0.0	4.012592
514	7	4	8	7	0.811613	0.191592	0.771315	0.033868	21.2	0.647059	6.7	0.0	2.498152
515	1	4	8	6	0.976774	0.499311	0.711622	0.201426	25.6	0.317647	4.0	0.0	0.000000
516	6	3	11	2	0.784516	0.006547	0.115867	0.019608	11.8	0.188235	4.5	0.0	0.000000

Nuestro modelo se basa en un polinomio donde a cada una de nuestras variables se le asigna una variable w_i y un bias b para poder tener un hiperplano que se acerque a los resultados originales. En nuestro experimento guardamos estos valores en una matriz para poder trabajar correctamente. Nuestra hipótesis quedaría de la siguiente manera:

$$H(X, W, b) = W^T * X + b$$

Para el entrenamiento de nuestro modelo primero generaremos un bias y un w aleatorio. Luego durante un número de iteraciones iremos mejorando nuestro modelo mediante el uso de derivadas. Primero debemos calcular nuestro error de la siguiente forma:

$$L = \sum_{i=0}^{n} (y_i - h(x_1))^2 / 2m$$

Ahora debemos buscar disminuir este error por cada iteración, para eso primero calculamos la derivada de nuestro error con respecto a todos los w_i y el b. Las derivadas de estos tienen la siguiente forma:

$$\partial L / \partial w_j = 1/m * \sum_{i=0}^{n} (y_i - h(x_1))(-w_j^i)$$

$$\partial L/\partial b = 1/m * \sum_{i=0}^{n}(y_i - h(x_1))(-1)$$

Luego con estas derivadas actualizamos los w_i y b con un α de la siguiente forma:

$$w_i = w_i - \alpha * \partial L/\partial w_j$$

$$b = b - \alpha * \partial L/\partial b$$

Finalmente actualizamos nuestro error y vamos a la siguiente iteración. Para nuestro experimento recibimos un parámetro *epocas* y un α para correr nuestro entrenamiento. En la imagen 2 podemos ver nuestras funciones para el entrenamiento.

Fig. 2. Funciones para el experimento

```
def hipotesis (X, W, b):
  return (np.dot(X,W)) + b
```

```
def computeCost(X,Y,W,b):
  m=len(Y)
  predictions = hipotesis(X,W,b)
  square_err=(predictions - Y)
  square_err = np.power(square_err,2)

  return 1/(2*m) * np.sum(square_err)
```

```
def training (Xt, Yt, Xv, Yv, W, b, alpha, epochs):
  m = len(Yt)
  J_history = []
  J_test_history = []
  for i in range (epochs):
    predictions = hipotesis (Xt, W, b)
    error = np.dot(Xt.transpose(),(predictions - Yt))
    errorb = np.sum(predictions - Yt)
    descent=alpha * 1/m * error
    descent=np.squeeze(np.asarray(descent)  )
    descentb=alpha * 1/m * errorb
    W = W - descent
    b-=descentb
    J_history.append(computeCost(Xt,Yt,W,b))
    J_test_history.append(computeCost(Xv,Yv,W,b))
  return W, J_history, J_test_history
```

A. *Resultados*

Corrimos nuestro modelo con 3 distintos valores de α: 0.001, 0.0001 y 0.01.

Para nuestro primer experimento obtenemos la gráfica de la figura 3 y observamos que trabajó bien con el modelo, y no encontramos ni overfitting ni underfitting.

En el segundo caso tenemos la gráfica de la figura 4, esta gráfica también trabaja bien debido a que el error termina disminuyendo al final, aunque el primer sigue mostrando mejores resultados. Finalmente en la figura 5 vemos nuestro último experimento. Esta gráfica muestra algunas complicaciones, podemos ver un caso de overfitting debido a que el error eventualmente aumenta cerca al final.

Finalmente podemos concluir que el modelo más óptimo es con un alpha 0.001 usado en el primer experimento.

Fig. 3. Experimento con alpha 0.001

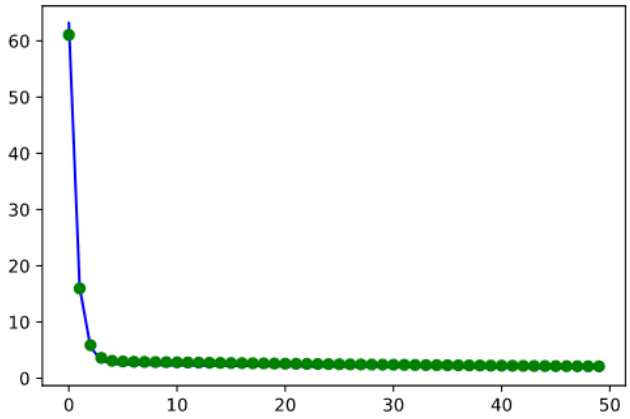
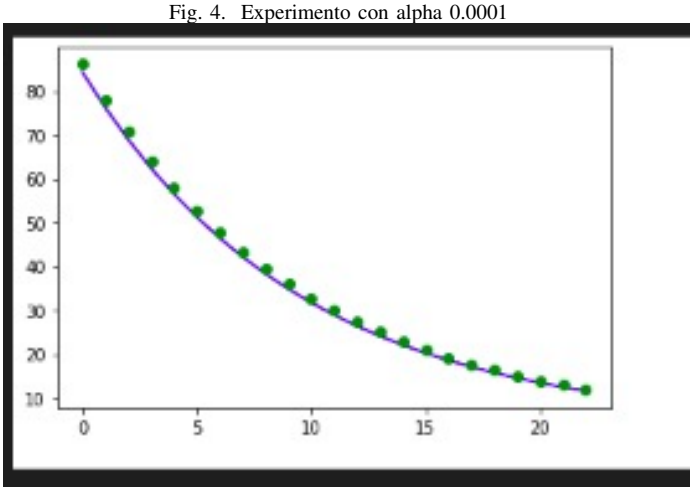


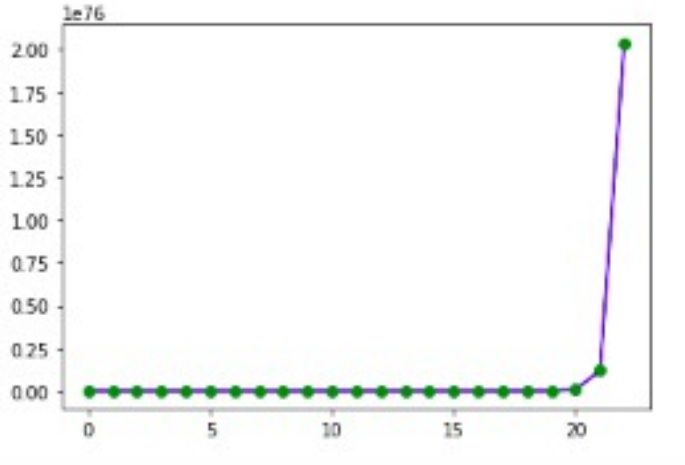
Fig. 4. Experimento con alpha 0.0001



IV. Conclusiones

- Hemos observado que la distribución normal es necesaria para poder hacer funcionar un modelo de regresión lineal, en este caso la columna 13 tenía un sesgo a la derecha por la mayor cantidad de ceros registrados en el campo de hectáreas quemadas. Pudimos observar que una función cuadrática reducía tambien el skewness pero la más eficaz fue la función logarítmica.
- Fue necesario normalizar las columnas 5,6,7,8 y 10 del dataset debido a que nos enfrentamos al problema de vanishing gradient. Es decir las gradientes se reducían a cero complicando el entrenamiento.
- Para este dataset la cantidad de épocas necesarias para lograr un correcto entrenamiento fue muy corta, aproximadamente 20 a 50 iteraciones. Sin embargo, si se aumenta el coeficiente de aprendizaje (alfa) no se lograba un correcto entrenamiento debido que se excedía del óptimo de la función costo.
- Por último, una optimización para el modelo es utilizar

Fig. 5. Experimento con alpha 0.01



un coeficiente de aprendizaje adaptativo. Para que vaya avanzando de forma más rapida al inicio y menos rápido cuando se acerca al óptimo de la función pérdida.