Project 1 - Artificial Intelligence

1st Alejandro Otero
dept. of Computer Science
University of Engineering and Technology
Lima, Perú
alejandro.otero@utec.edu.pe

2nd Angélica Sánchez
dept. of Computer Science
University of Engineering and Technology
Lima, Perú
angelica.sanchez@utec.edu.pe

I. Introducción

Dentro del campo de Inteligencia Artificial existen diversos métodos para entrenar un algoritmo, de modo que este pueda aproximar ciertos valores para obtener un resultado a partir de estimaciones. En esta oportunidad la técnica utilizada es regresión lineal, pero hay que tener en cuenta que existe la regresión lineal simple y la múltiple, donde la diferencia entre ambas es la cantidad de variables independientes utilizadas. En este informe se explicará un ejemplo de la regresión lineal múltiple, también llamada multivariada, así como algunos experimentos realizados para entender su comportamiento.

El dataset utilizado en este proyecto cuenta con 12 atributos representan las características del suelo, los cuales serán utilizadas como las variables independientes (x) y 1 atributo adicional correspondiente al área del terreno que podría incendiarse dependiendo de los atributos del suelo, el cual representará la variable dependiente (y).

II. Explicación

Para trabajar con una regresión se tiene una estructura la cual es la siguiente:

- Preparar el dataset a utilizar
- Una función hipótesis
- La función error del modelo
- Las derivadas de la función error con respecto a **b** y **w**
- La función actualizar para cambiar los valores de **b** y **w**
- La función principal, en la cual se ejecutan todas las llamadas a las funciones ya mencionadas, así como asignar un valor aleatorio a **b**, a **w**, a **alpha**, a **k** a partir del número de atributos y a **epochs** para determinar la cantidad de iteraciones.

A. Función h(x, w, b)

El modelo de regresión se planteó como una función lineal:

$$f(x) = w * x + b \tag{1}$$

En la cual tanto **w** como **x** representan matrices, **w** una matriz de `1 x k` y **x** una matriz de `size x k`, donde `size` corresponde al tamaño del subgrupo de entrenamiento y `k` corresponde a la cantidad de variables independientes. Para poder multiplicarlas las interpretábamos como vectores y aplicábamos producto punto.

B. Función derivate(x, y, w, b, k)

En esta función se calculan las derivadas de la función de error con respecto a **b** y a **w**.

$$\frac{dL}{db} = (\sum_{i=0}^{n} y_i - h(x))(-1) \tag{2}$$

$$\frac{dL}{dw_j} = (\sum_{i=0}^{n} y_i - h(x))(-x_j^i) \tag{3}$$

C. Función update(w, b, dw, db, alpha, k)

Se actualizan los valores de **w** y **b** para entrenar el algoritmo a través de la siguiente ecuación:

$$w = w - \alpha * dw \tag{4}$$

$$b = b - \alpha * db \tag{5}$$

Hay que tener en cuenta que esto se hace para todos los valores dentro de la lista **w**. En el caso de **b** es la misma lógica, pero solo una vez, ya que este es un único valor.

D. Función error(x, y, w, b)

Se evalúa la ecuación de error:

$$Error = \frac{\sum_{i=0}^{n}(y_i - h(x))^2}{2} \tag{6}$$

E. Función regression(x_train, y_train, x_validation, y_validation, alpha, epochs, k)

En esta función se realizan todas las llamadas a las funciones ya mencionadas anteriormente y se repite este proceso **epochs** veces. Asimismo, inicialmente se utilizan los subgrupos de entrenamiento y validación, ya que a partir de estos podemos identificar si el modelo está teniendo los resultados esperados. Hay que tener en cuenta que estos subgrupos se dividen en 70% y 20%respectivamente, donde el 10% restante corresponde al subgrupo de testeo. Además, tanto **b** y **w** se generan de manera aleatoria. Luego, tanto **alpha** como **epochs** pueden ser números aleatorios, pero en el caso de **alpha** debe ser un número pequeño y en el caso de **epochs** debe ser un número grande.

Finalmente, **k** corresponde a la cantidad de atributos/variables independientes (input **x**). Las llamadas a las funciones se realizan en el siguiente orden:

1) derivate()
2) update()
3) error() para el primer subgrupo (entrenamiento)
4) error() para el segundo subgrupo (validación)
5) Se guardan ambos errores en dos listas en cada iteración para luego poder graficarlos

III. Experimentos

1) Para comparar las curvas del error de entrenamiento y del error de validación en cada iteración, se realizaron 6 pruebas (las cuales se pueden apreciar en las figuras Fig.1, Fig.2, Fig.3, Fig.4, Fig.5 y Fig.6) con 6 valores aleatorios de `alpha` para una cantidad de 100 épocas.
2) Para comparar el error de entrenamiento, el error de validación y el error de testeo se realizó una prueba (la cual se puede apreciar en la figura Fig.7) con un valor `alpha = 0.05` para una cantidad de 100 épocas.

A. Resultados

A partir de los gráficos se puede apreciar que nuestras curvas tienen un comportamiento similar, así como una distancia pequeña lo que nos da a entender que se ha aplicado el ajuste ideal al modelo planteado.

1) Basándose en lo observado en las primeras pruebas, donde se comparan las curvas de error de entrenamiento y de error de validación, vemos que estas si logran ajustarse de manera correcta. Sin embargo, en las figuras Fig.2, Fig.4 y Fig.5, se puede apreciar que en la parte inicial de las curvas se genera un underfitting y eso se debe al valor de **alpha**.
   A diferencia de las otras figuras (Fig.1, Fig.3 y Fig.6), estas cuentan con un valor de **alpha** más grande, pues la cantidad de épocas son las mismas. Asimismo, hay que tener en cuenta que esto sucede, debido a que **alpha** al ser pequeño provoca que la curva llegue al valor óptimo lentamente, es decir, acercarse al error mínimo.
   Podemos observar en las figuras Fig.8 y Fig.9 cómo se comportan las curvas utilizando un valor más pequeño y más grande para **alpha**, respectivamente. Esto demuestra que, en efecto, **alpha** afecta en la velocidad ajuste de las curvas.
2) Basándose en lo observado en la última prueba correspondiente a la comparación del error de entrenamiento, el error de validación y el error de testeo, vemos que la curva de entrenamiento y la de testeo son casi iguales. Esto podría deberse a distintas razones, en el mejor de los casos nuestro modelo fue construido casi a la perfección, por ende al momento de realizar el testeo no existe una diferencia de error. Sin embargo, puede deberse también a que los valores del subgrupo de testeo sean muy similares a los del subgrupo de entrenamiento.

IV. Conclusiones

En conclusión, luego de haber realizado diversos experimentos para entender el comportamiento de las curvas y ver si se generaba un overfitting o un underfitting, entendimos que el causante de que se dé o no es **alpha**, puesto que justamente las gráficas que tenían un valor bajo de esa variable se acercaban al error mínimo lentamente, en otras palabras, generaban un underfitting al inicio por demorarse en llegar el valor óptimo. Caso contrario, las gráficas con un valor de **alpha** grande logran llegar a un valor óptimo rápidamente, pues se puede apreciar en los experimentos que se acercan al error mínimo en menos iteraciones. No obstante, hay que tener en cuenta que tener un valor de **alpha** muy grande tampoco es beneficioso, pues se puede dar el caso que nunca alcance el valor óptimo por más que descienda velozmente.

V. Anexos

Link al repositorio de GitHub:
https://github.com/Oteranga/Proyecto1_IA

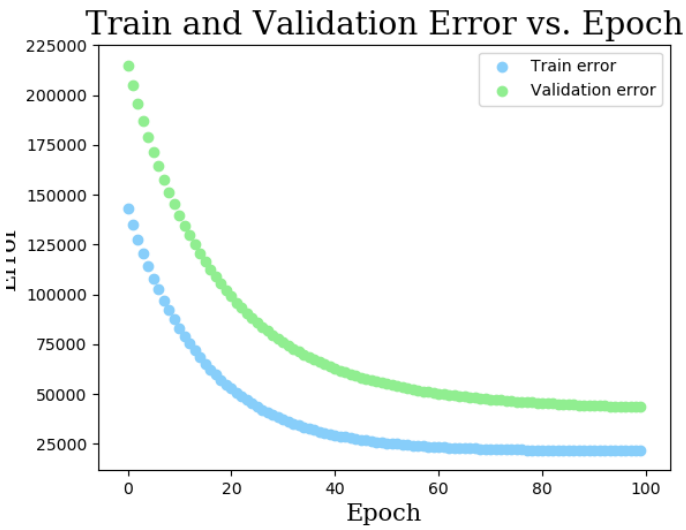

Fig. 1. Plot para un valor de `alpha = 0.0334`

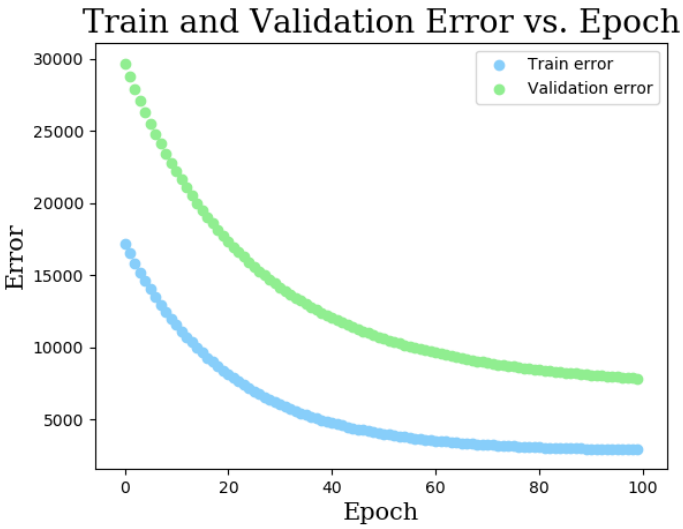

Fig. 2. Plot para un valor de `alpha = 0.0246`

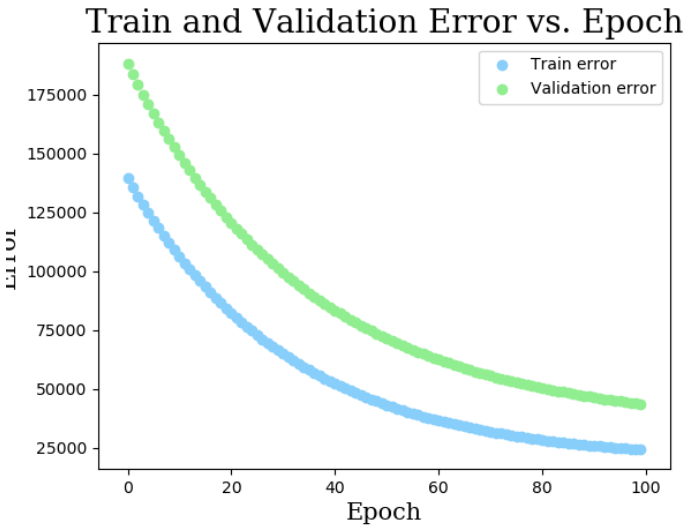

Fig. 4. Plot para un valor de `alpha = 0.016`

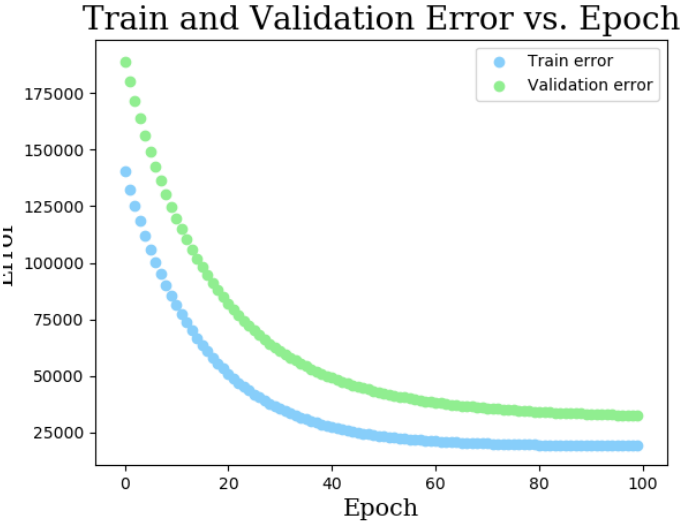

Fig. 3. Plot para un valor de `alpha = 0.0327`

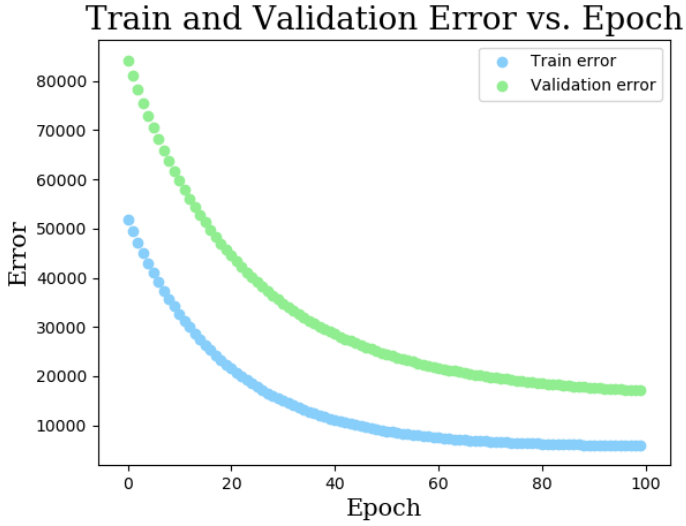

Fig. 5. Plot para un valor de `alpha = 0.0262`

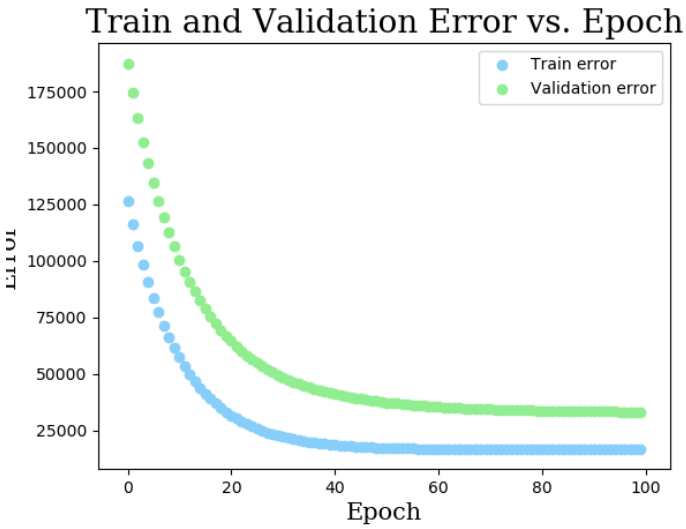

Fig. 6. Plot para un valor de `alpha = 0.0483`

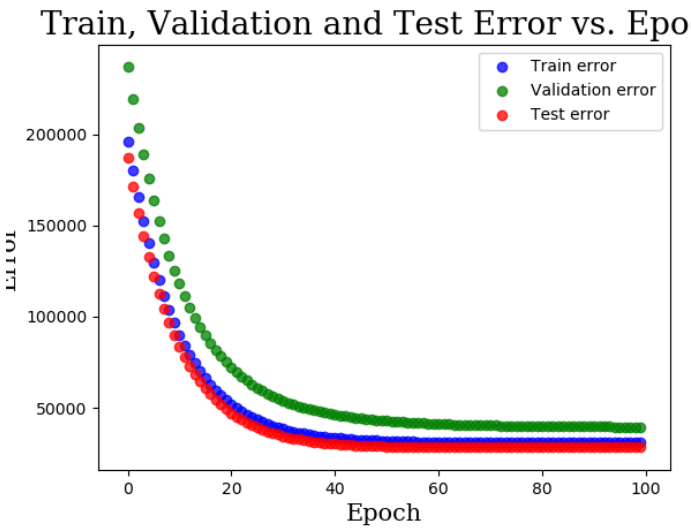

Fig. 7. Plot para un valor de `alpha = 0.05`

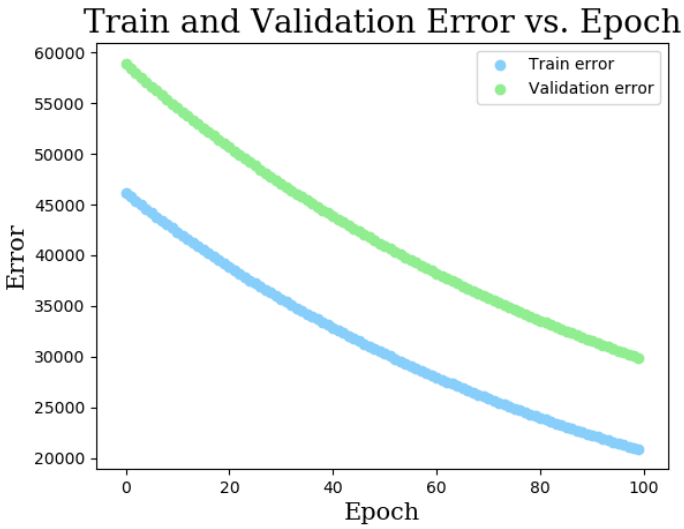

Fig. 8. Plot para un valor más pequeño de `alpha = 0.005`

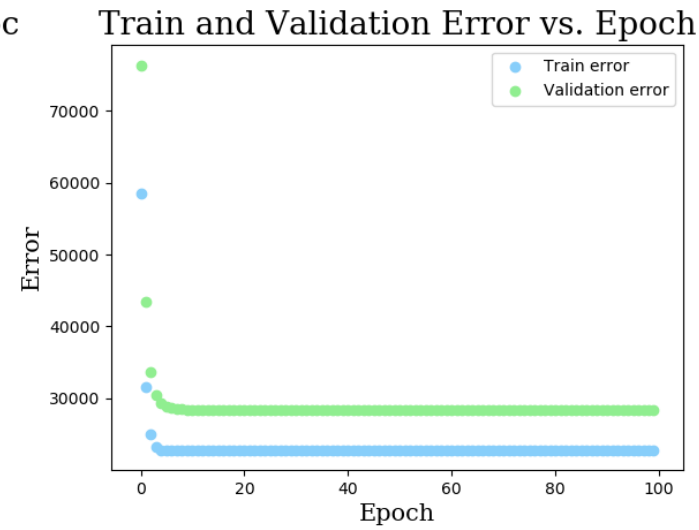

Fig. 9. Plot para un valor más grande de `alpha = 0.5`