Universidad Galileo

Algoritmos en la Ciencia de Datos

Nombre: Rodrigo Rafael Chang Papa

Carné: 19000625

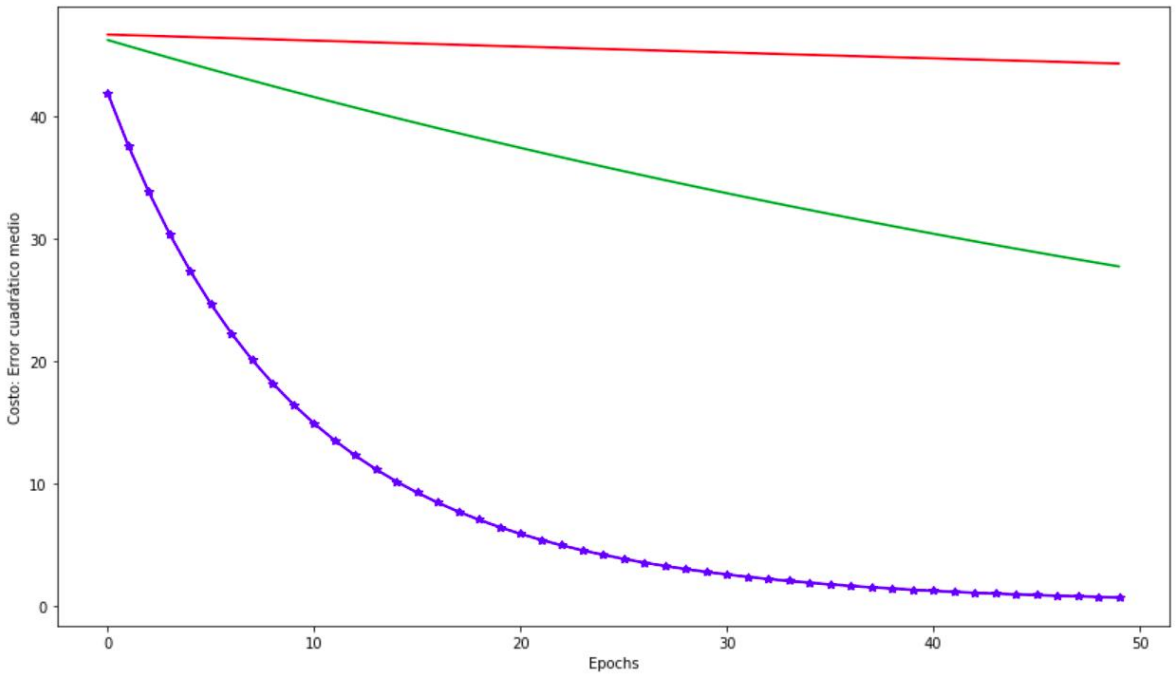
Regresiones lineales vía GD, SGD y Minibatch

Parte 1 - Solución cerrada

Para este ejercicio se implementó la solución cerrada al problema de mínimos cuadrados ordinarios (MCO). De acuerdo con las propiedades de los estimadores de MCO, estos son los mejores estimadores lineales e insesgados para el problema de regresión lineal. Se observó que estos estimadores fueron los más cercanos a los verdaderos parámetros.

En la práctica no se utiliza este método de estimación debido a que la matriz $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ es una matriz de $\mathrm{n} \times \mathrm{n}$, donde n representa el número de variables explicativas en el modelo y puede ser muy grande para obtener la matriz inversa de forma eficiente.

Parte 2 - Gradient descent



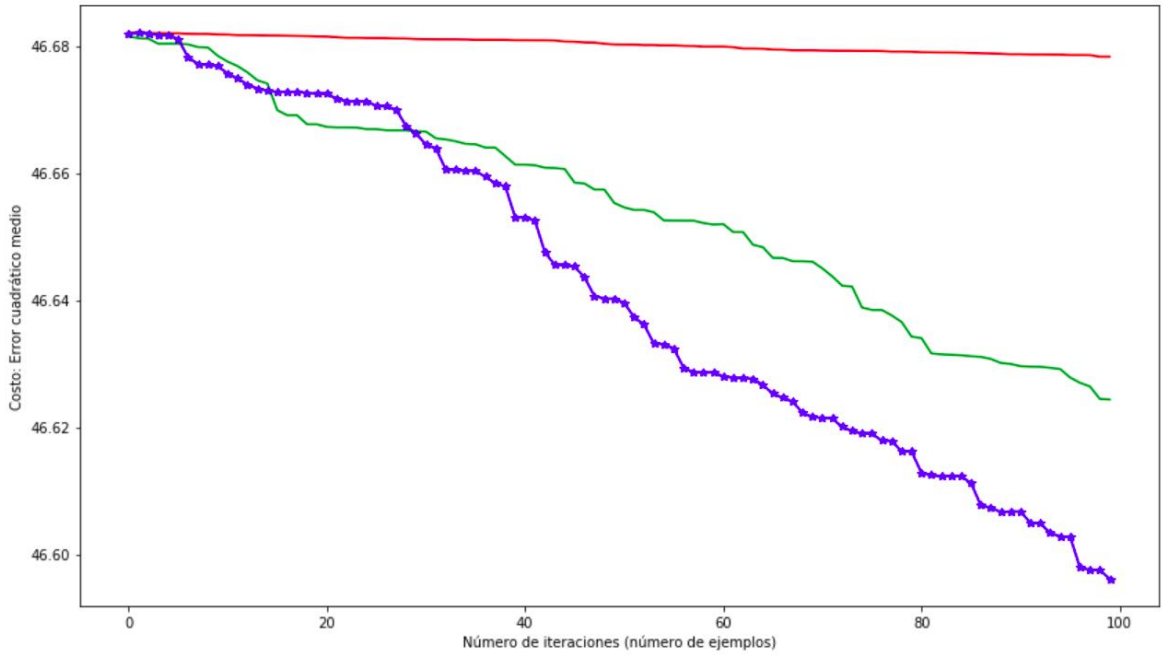
Gráfica con los 3 tipos de *step size*: el azul corresponde al learning rate más grande ($\alpha = 0.05$).

Para este problema de minimización, aplicando GD se observó que mientras mayor era el tamaño del *step size,* la convergencia hacia el mínimo de la función se alcanzaba más rápidamente. Sin embargo, si el tamaño de paso era demasiado grande, la solución empezaba a oscilar y divergía.

Aunque hubieron varios *step sizes* que permitieron alcanzar minimizar aceptablemente la función, se encontró que con un $\alpha = 0.05$ y 100 *epochs* de entrenamiento se obtenían buenos resultados.

Parte 3 - Stochastic Gradient Descent

Gráfica de algunas iteraciones utilizando el algoritmo de Stochastic Gradient Descent. La línea azul representa el entrenamiento con $\alpha = 0.01$ y 2 epochs de entrenamiento.

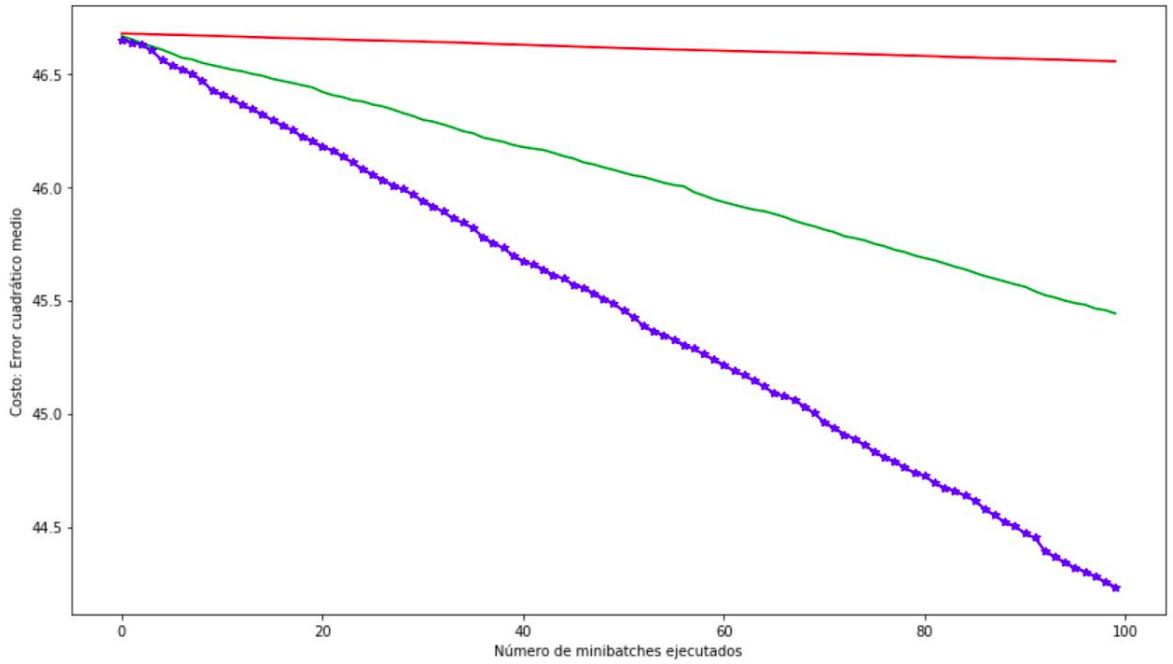


Como se observa, debido a que el entrenamiento de la función se realiza ejemplo por ejemplo, la convergencia es más lenta por iteración y aunque es estrictamente decreciente en todas las iteraciones iteraciones, en algunas es díficil apreciar que la iteración haya tenido efecto minimizando la función.

En este caso, también se observó que con tamaños de paso grandes, por ejemplo con $\alpha = 0.01$ se alcanzaba la convergencia más rápidamente, pero con este método requería más iteraciones de todas formas.

Parte 4 - Minibatch Gradient Descent

Gráfica de algunas iteraciones (de *minibatches*) utilizando el algoritmo de Minibatch Gradient Descent. La línea azul representa el entrenamiento con $\alpha = 0.01$, 100 epochs de entrenamiento y 100 ejemplos por *minibatch*.



En este caso, se observó que con un *step size* moderado (por ejemplo, $\alpha = 0.01$) se obtenían buenos resultados. En general el algoritmo se comportó bien incluso con valores mayores del step size.

En cuanto al tamaño del batch, se observó que a mayor número de ejemplos por minibatch, el comportamiento de cada paso era más eficiente en disminuir el valor de la función de costo. La intuición detrás de esto es que la función de costo es la suma de los costos individuales generados por cada una de las observaciones en el conjunto de datos, y por lo tanto, al tomar más ejemplos para actualizar los parámetros, nos acercamos más rápidamente a los minimizadores de la función objetivo.

Se observó que una combinación que dió buenos resultados es la de $\alpha = 0.01$ y 100 ejemplos por minibatch.

Parte 5 - Comparación

A continuación, se hará una clasificación de los diferentes métodos de acuerdo a diferentes criterios: el valor óptimo de la función de costo, el número de iteraciones y una medida del error en los parámetros obtenidos respecto a los verdaderos parámetros (se utilizará la distancia euclidiana entre el vector de parámetros betas).

Método	**Función de costo f^***	**Iteraciones totales**	**Tiempo**	**Distancia**	**Posición**
Solución cerrada	0.12065	-	0.07294	0.1753	**1**
GD	0.14194	100 epochs de 1000 ejemplos c/u	0.05608	0.3123	**2**
SGD	6.74515	100 epochs de 1000 ejemplos c/u	21.6267	4.0513	**4**
MBGD	0.14386	500 epochs de 500 ejemplos c/u	0.54958	0.3220	**3**

En este caso, como se observa, el método de solución cerrada fue el campeón en tiempo, valor final de la función objetivo, mejor estimación de parámetros y número de iteraciones. Sin embargo, como se mencionaba anteriormente, si el problema de estimación se vuelve masivo, por ejemplo, con un número de parámetros 1000 veces mayor, posiblemente tardaría bastante más debido a que aumentaría la complejidad del cómputo de la matriz inversa.

Por otra parte, en este ranking queda mejor el algoritmo de GD que el de MBGD. Sin embargo, en la práctica es más utilizado el de MBGD, debido a que si el conjunto de datos de entrenamiento es muy grande, cada una de las iteraciones puede ser muy costosa en cómputo, por lo que se toman porciones más pequeñas (minibatches) para llevar a cabo el entrenamiento. Para este ejercicio cada uno de los epochs se implementó con loops, lo que hace más lento el cómputo, mientras que el gradiente se implementó vectorizado, por lo

que la diferencia en el tiempo se debió más que todo al mayor número de iteraciones. Por lo que el segundo y tercer lugar queda muy disputado entre GD y MBGD.

Finalmente, como se observa, el algoritmo de SGD fue el más lento debido a que se consideró un ejemplo a la vez por iteración del ciclo. Además, de los diferentes algoritmos, fue el que generó parámetros más lejanos respecto a los verdaderos, debido a que requería todavía más iteraciones para seguir minimizando la función de costo. En este caso, este algoritmo podría ser útil si se necesita retomar el entrenamiento con algunos nuevos ejemplos, o si el entrenamiento se tiene que llevar a cabo “en vivo”, conforme se generan nuevas observaciones.