Descenso de Gradiente Estocástico
Proyecto de Investigación
CA0504: Optimización, Prof. Álvaro Guevara

Michael Abarca
Jennifer Acuña
José Emmanuel Chacón

Julio 2, 2019

1.1. Introducción

En este trabajo se presentan métodos de optimización basados en evaluación de gradiente, específicamente el Descenso de Gradiente y Descenso de Gradiente Estocástico; dando detalles de los algorítmos y soporte teórico a su convergencia bajo ciertas condiciones. Posteriormente se considera un problema de aplicación de clasificación, y se analizan los resultados obtenidos al calibrar modelos de Máquinas de Soporte Vectorial y Regresión Logística.

Se utilizan los métodos mencionados para minimizar las funciones de costo de cada modelo, y se realizan escenarios de prueba de rendimiento entre diferentes variaciones de los métodos y de parámetros de los mismos. Estas aplicaciones se realizan en *Matlab* usando la librería *SGDLibrary* de PhD Hiroyuki Kasai, y se comparan resultados obtenidos con otras librerías fuera de *Matlab*.

1.2. Descenso de Gradiente

Recordemos que dada una función $f : D \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$ diferenciable en un punto $\boldsymbol{x}$ en D, se define el *gradiente* como el vector que contiene las primeras derivadas parciales

$$\nabla f(\boldsymbol{x}) := \left(\frac{\partial f(\boldsymbol{x})}{\partial\, x_1}, \frac{\partial f(\boldsymbol{x})}{\partial\, x_2}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial\, x_n} \right).$$

Geométricamente, se trata de un vector normal al hiperplano tangente a una hipersuperficie de la forma $f(\boldsymbol{x}) = 0$. Recordemos además que el gradiente tiene la propiedad de apuntar en la dirección de máximo crecimiento de la función $f(\boldsymbol{x})$. Por otro lado, los puntos $\boldsymbol{x}$ que satisfacen la ecuación $\nabla f(\boldsymbol{x}) = 0$ son fuertes candidatos a ser óptimos de una función. Estas razones nos indican que es posible utilizar el gradiente en problemas de optimización, ya que por medio de éste se pueden llegar a obtener los óptimos deseados.

Consideremos el problema de optimización que consiste en minimizar una función objetivo $f(\boldsymbol{x})$ para $\boldsymbol{x}$ en $\mathbb{R}^p$.

Métodos numéricos típicos toman como punto de partida un valor inicial $\boldsymbol{x}_0$ el cual van actualizando iterativamente hasta lograr llegar al mínimo o a un valor suficientemente cercano al mínimo. Generalmente las sucesivas actualizaciones de $\boldsymbol{x}_0$ se obtienen por medio de una dirección $\boldsymbol{v}$, de modo que si $f(\boldsymbol{x} + \alpha\, \boldsymbol{v}) < f(\boldsymbol{x})$ entonces se actualiza $\boldsymbol{x}$ como $\boldsymbol{x} := \boldsymbol{x} + \alpha\, \boldsymbol{v}$.

Al valor $\alpha > 0$ se le llama *learning rate* y define el tamaño del paso en cada iteración del algoritmo y al vector $\boldsymbol{v}$ se le llama *dirección descendiente*. Ahora el problema consiste en determinar la forma de $\boldsymbol{v}$ de modo que las actualizaciones de $\boldsymbol{x}_0$ se aproximen al mínimo. Este proceso es lo que en general se conoce como *line search*.

Dadas las propiedades del gradiente discutidas al inicio de esta sección, una forma de elegir esta dirección descendiente es utilizando el negativo del gradiente, $-\nabla f(\boldsymbol{x})$.

Con la elección anterior para la dirección descendiente el método de line search toma el nombre de Descenso de Gradiente. Por tanto el Descenso de Gradiente es un método iterativo que busca solucionar el problema de minimización actualizando $\boldsymbol{x}$ en cada iteración de la siguiente forma,

$$\boldsymbol{x} := \boldsymbol{x} - \alpha\, \nabla f(\boldsymbol{x}).$$

Veamos una justificación más directa de la elección de dirección de gradiente $\boldsymbol{v} = -\nabla f(\boldsymbol{x})$. Sea $g(\alpha) = f(\boldsymbol{x} + \alpha\, \boldsymbol{v})$, que es una función de dominio real. Utilizando la expansión de Taylor de orden 1 alrededor de 0, se tiene

$$g(\alpha) = g(0) + g'(0)\, \alpha + O(h^2).$$

Note que:

(a) $g(0) = f(\boldsymbol{x})$.

(b) $g'(\alpha) = \sum_{i=1}^{p} f_i'(\boldsymbol{x} + \alpha\, \boldsymbol{v})\, \boldsymbol{v}_i$, entonces $g'(0) = \sum_{i=1}^{p} f_i'(\boldsymbol{x})\, \boldsymbol{v}_i = (\nabla f(\boldsymbol{x}))^T\, \boldsymbol{v}$.

Por tanto se tiene que

$$f(\boldsymbol{x} + \alpha\, \boldsymbol{v}) = f(\boldsymbol{x}) + \alpha\, (\nabla f(\boldsymbol{x}))^T\, \boldsymbol{v} + O(h^2).$$

Luego, tomando $\boldsymbol{v} = -\nabla f(\boldsymbol{x})$ obtenemmos

$$f(\boldsymbol{x} + \alpha\, \boldsymbol{v}) = f(\boldsymbol{x}) - \alpha\, \|\nabla f(\boldsymbol{x})\|^2 + O(h^2).$$

Entonces para valores pequeños de α se cumple que

$$f(\boldsymbol{x} + \alpha\, \boldsymbol{v}) < f(\boldsymbol{x}).$$

El algoritmo del Descenso de Gradiente es el siguiente.

1. Seleccione una learning rate $\alpha > 0$ y parámetros iniciales $\boldsymbol{x}_0$.
2. Actualiza $\boldsymbol{x}$ de la siguiente forma hasta obtener un mínimo aproximado (iteraciones)

$$\boldsymbol{x} := \boldsymbol{x} - \alpha \nabla f(\boldsymbol{x}).$$

Note que cada iteración implica el usar todos los datos, es decir una sola pasada sobre los datos representa una sola actualización de los parámetros.

Convergencia del Método de Descenso de Gradiente

A continuación se muestra que el Descenso de Gradiente converge al mínimo global para funciones Lipschitz convexas.

Teorema 1.1. *Sea $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ una función convexa y diferenciable, con gradiente Lipschitz continuo con constante $L > 0$, i.e. tenemos que $||\nabla f(x) - \nabla f(y)||_2 \leq L||x - y||_2$ para cualquier x, y. Entonces si se efectúa el método de Descenso de Gradiente k veces con un paso fijo $\alpha \leq \frac{1}{L}$ se obtiene una solución $x^{(k)}$ que satisface*

$$f(x^{(k)}) - f(x^*) \leq \frac{||x^{(0)} - x^*||_2^2}{2\alpha k}$$

donde $f(x^*)$ es el valor óptimo. Intuitivamente, esto garantiza que el Descenso de Gradiente converge y que su velocidad de convergencia es $O(1/k)$.

Solución. Como ∇f es Lipschitz continua con constante L, entonces $\nabla^2 f(x) - LI$ es una matriz semidefinida positiva, luego

$$\begin{aligned} f(y) &\leq f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}\nabla^2 f(x)||y-x||_2^2 \\ &\leq f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}L||y-x||_2^2 \end{aligned}$$

Actualizando el descenso de gradiente, se tiene que $y = x^+ = x - \alpha\nabla f(x)$. Luego, se obtiene

$$\begin{aligned} f(x^+) &\leq f(x) + \nabla f(x)^T(x^+ - x) + \frac{1}{2}L||x^+ - x||_2^2 \\ &= f(x) + \nabla f(x)^T(x - \alpha\nabla f(x) - x) + \frac{1}{2}L||x - \alpha\nabla f(x) - x||_2^2 \\ &= f(x) - \nabla f(x)^T\alpha\nabla f(x) + \frac{1}{2}L||\alpha\nabla f(x)||_2^2 \\ &= f(x) - \alpha||\nabla f(x)||_2^2 + \frac{1}{2}L\cdot\alpha^2||\nabla f(x)||_2^2 \\ &= f(x) - \left(1 - \frac{L}{2}\alpha\right)\alpha||\nabla f(x)||_2^2 \end{aligned}$$

Usando $t \leq \frac{1}{L}$, entonces

$$-\left(1 - \frac{L}{2}\alpha\right) \leq \frac{1}{2} - 1 = -\frac{1}{2}$$

así se obtiene que

$$f(x^+) \leq f(x) - \frac{1}{2}\alpha||\nabla f(x)||_2^2$$

note que como $\frac{1}{2}\alpha||\nabla f(x)||_2^2 > 0$(a menos que $\nabla f(x) = 0$), la desigualdad anterior implica que la función objetivo decrece con cada iteración del Descenso de Gradiente hasta que alcance el valor óptimo $f(x) = f(x^*)$, en cuyo caso $\nabla f(x) = 0$. Esto es válido siempre que α sea lo suficientemente pequeño, i.e. $\alpha \leq 1/L$.

La segunda parte de la prueba consiste en acotar el valor de $f(x^+)$ en términos del valor óptimo. Como f es convexa, se puede escribir

$$\begin{aligned} f(x^*) &\geq f(x) + \nabla f(x)^T(x^* - x) \\ \Rightarrow f(x) &\leq f(x^*) + \nabla f(x)^T(x - x^*) \end{aligned}$$

Luego,

$$
\begin{aligned}
f(x^+) &\leq f(x^*) + \nabla f(x)^T(x - x^*) - \frac{\alpha}{2}||\nabla f(x)||_2^2 \\
\Rightarrow f(x^+) - f(x^*) &\leq \frac{1}{2\alpha}\left(2\alpha\nabla f(x)^T(x - x^*) - \alpha^2||\nabla f(x)||_2^2\right) \\
&\leq \frac{1}{2\alpha}\left(2\alpha\nabla f(x)^T(x - x^*) - \alpha^2||\nabla f(x)||_2^2 - ||x - x^*||_2^2 + ||x - x^*||_2^2\right) \\
&\leq \frac{1}{2\alpha}\left(||x - x^*||_2^2 - ||x - x^* - \alpha\nabla f(x)||_2^2\right) \\
&= \frac{1}{2\alpha}\left(||x - x^*||_2^2 - ||x^+ - x^*||_2^2\right)
\end{aligned}
$$

Sumando sobre cada iteración x^+ del descenso de gradiente, se obtiene

$$
\begin{aligned}
\sum_{i=1}^{k} f(x^{(i)}) - f(x^*) &\leq \sum_{i=1}^{k} \frac{1}{2\alpha}\left(||x^{(i-1)} - x^*||_2^2 - ||x^{(i)} - x^*||_2^2\right) \\
&\leq \frac{1}{2\alpha}\left(||x^{(0)} - x^*||_2^2 - ||x^{(k)} - x^*||_2^2\right) \\
&\leq \frac{1}{2\alpha}\left(||x^{(0)} - x^*||_2^2\right)
\end{aligned}
$$

Finalmente, se concluye que

$$
\begin{aligned}
f(x^{(k)}) - f(x^*) &\leq \frac{1}{k}\sum_{i=1}^{k} f(x^{(i)}) - f(x^*) \\
&\leq \frac{||x^{(0)} - x^*||_2^2}{2\alpha k}
\end{aligned}
$$

□

Problemas del Método

El método de Descenso de Gradiente tiene una serie de desventajas o desafios que afectan su rendimiento.

(a) **Mínimos Locales:** El método una vez que el método llega a un mínimo local la regla de actualización de los parámetros deja de actualizar, pues el gradiente es el vector 0 en dicho punto. Por tanto el método es suceptible es a caer en mínimos locales que no pueden ser tan buenos.

(b) **Puntos Silla:** Similar a lo anterior, una vez que el método llega a un punto silla se alcanza la convergencia pues el Gradiente es el vector 0. En este caso el problema es aún más grave pues los puntos sillas no son minimos de ningún tipo.

(c) **Valor Inicial:** Como casi en cualquier método de optimización el valor inicial puede representar una dificultad. En este método en particular la selección del valor inicial lo puede llevar directamente a contrar el mínimo global, o bien a encontrar uno local o bien caer en un punto silla. Por lo que el valor inicial es un parámetro sensible del modelo.

(d) **Grandes Tablas de Datos:** Cada paso o iteración en el método involucra usar toda la tabla de datos pues implica calcular el gradiente de la función de costo la cual está definida como un promedio de los costos individuales de cada fila de la tabla. Dicho cálculo puede ser muy lento computacionalmente para tablas de varios millones de filas, pues cargar en memoria dichas tablas para hacerlos cálculos puede ser muy pesado y poco práctico.

La siguiente figura muestra una representación en 3 dimensiones de la superficie de una función de costo para una red neuronal. Podemos observar como los mínimos locales, los puntos silla y el valor inicial pueden ser un gran problema para el método de Descenso de Gradiente.

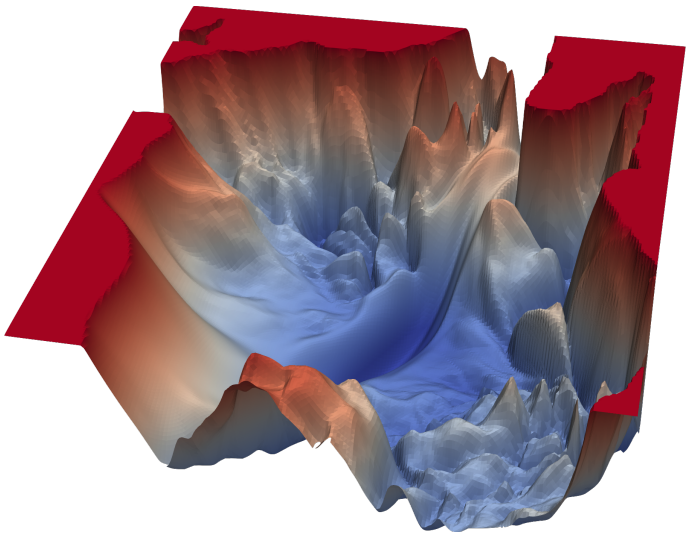


Figura 1.1: Función de Costo para una Red Neuronal

1.3. Descenso de Gradiente Estocástico

El método de Descenso de Gradiente Estocástico es una modificación y una alternativa al método clásico de Descenso de Grandiente, que permite superar los desafíos vistos en la sección anterior para el método clásico, ésto lo logra mediante el uso de la aleatoriedad.

La idea fundamental detrás del Descenso de Gradiente Estocástico es usar un estimador del gradiente en cada iteración. El uso de una estimación y la aleatoriedad en ella es lo que le permite escapar de los mínimos locales y puntos silla, y seguir avanzado en la búsqueda del mínimo global o mínimos locales de buena calidad.

Nuevamente considere el problema de minimizar una función objetivo $f(\boldsymbol{x})$ para $\boldsymbol{x}$ en $\mathbb{R}^p$. El Descenso de Gradiente Estocástico es un método iterativo que busca solucionar el problema de minimización actualizando $\boldsymbol{x}$ en cada iteración de la siguiente forma,

$$\boldsymbol{x} := \boldsymbol{x} - \alpha\, \boldsymbol{v},$$

donde $\boldsymbol{v}$ es un vector aleatorio y un estimador insesgado de $\nabla f(\boldsymbol{x})$, es decir $\mathbb{E}[\boldsymbol{v}] = \nabla f(\boldsymbol{x})$. Note la similitud con el método de Descenso de Gradiente, solo que ahora en lugar de usar el gradiente usamos un estimador insesgado de éste.

El valor $\alpha > 0$ es la *learning rate* y define el tamaño del paso en cada iteración del algoritmo, puede ser un valor fijo o decrecer en cada iteración. El vector $\boldsymbol{v}$ es la *dirección descendiente estocástica*, y es la aproximación del gradiente que se emplea. Ahora el problema se reduce a encontrar una expresión adecuada para este estimador.

Supongamos que la función objetivo está dada por la suma finita de un número n de funciones

$$f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}).$$

Desde el punto de vista aplicado f es la función de costo total, f_i el costo individual de la fila i de la tabla de datos y n es la cantidad de filas en la tabla.

En cada iteración en lugar de calcular el gradiente $\nabla f(\boldsymbol{x})$ el método de forma aleatoria toma un índice i del conjunto $\{1, ..., n\}$ y calcula $\nabla f_i(\boldsymbol{x})$, que usa como estimador insesgado de $\nabla f(\boldsymbol{x})$. Veamos que en efecto es insesgado

$$\mathbb{E}_i[\nabla f_i(\boldsymbol{x})] = \sum_{i=1}^{n} \nabla f_i(\boldsymbol{x})\, p(i) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}) = \nabla\Big(\frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x})\Big) = \nabla f(\boldsymbol{x}).$$

Usando el estimador anterior el método actualiza $\boldsymbol{x}$ en cada iteración de la siguiente manera,

$$\boldsymbol{x} := \boldsymbol{x} - \alpha \nabla f_i(\boldsymbol{x}).$$

En una generalización del método llamada Mini Batch SGD, en cada iteración se toma de forma aleatoria una colección $\mathcal{B}$ de índices del conjunto $\{1, ..., n\}$, y usa

$$\nabla f_{\mathcal{B}}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \nabla f_i(\boldsymbol{x})$$

como estimador insesgado de $\nabla f(\boldsymbol{x})$. Y así actualiza $\boldsymbol{x}$ como

$$\boldsymbol{x} := \boldsymbol{x} - \alpha \nabla f_{\mathcal{B}}(\boldsymbol{x}).$$

En la generalización a $\mathcal{B}$ se le llama *mini batch* o simplemente *batch* y a $|\mathcal{B}|$ se le llama *batch size*. Note que en el planteamiento original $\mathcal{B} = \{i\}$ y $|\mathcal{B}| = 1$.

El número de iteraciones del método estará definido por el batch size, y está dado por $\frac{n}{|\mathcal{B}|}$ (suponiendo que $|\mathcal{B}|$ es un divisor de n, si no fuera el caso se usan $\lceil \frac{n}{|\mathcal{B}|} \rceil$ iteraciones). Entonces para un n fijo entre mayor sea el batch size menor cantidad de iteraciones y por tanto menor tiempo de ejecución. El caso $|\mathcal{B}| = 1$ es el más pesado computacionalmente, por lo que en la práctica suelen dar buenos resultados $|\mathcal{B}| = 32, 64$ o 128.

En la práctica se suele especificar el batch size a usar y a partir de dicho número se divide la tabla en batches disjuntos que son del mismo tamaño $|\mathcal{B}|$ (excepto quizá el último que puede tener un tamaño menor que $|\mathcal{B}|$). Por tanto, una vez que se completan todas las iteraciones habremos dado una pasada compela a los datos, en el sentido de que cada una

de las filas habrá sido usada exactamente 1 vez en el algoritmo. A esta pasada completa por los datos le llamamos *epoch*. El método puede usar varias epochs, lo cual permite acercar más los parámetros al óptimo; sin embargo usar pocas puede causar underfitting y usar muchas puede causar overfitting.

La motivación del Mini Batch SGD surge del hecho que el planteamiento original con batch size de 1 puede ser muy pesado computacionalmente con largos tiempos de ejecución y muchas veces puede no ser viable. El hecho de usar un batch size mayor a 1 puede reducir drásticamente estos tiempos sin comprometer la calidad de los resultados.

El algoritmo para el planteamiento original del Descenso de Gradiente Estocástico es el siguiente.

1. Seleccione una learning rate $\alpha > 0$ y parámetros iniciales $\boldsymbol{x}_0$.
2. Repetir hasta obtener un mínimo aproximado (epochs).
   a) Considere una permutación aleatoria A del conjunto $\{1, \ldots, n\}$
   b) De forma secuencial para cada i en A se actualiza $\boldsymbol{x}$ de la siguiente forma (iteraciones)
   $$\boldsymbol{x} := \boldsymbol{x} - \alpha \nabla f_i(\boldsymbol{x}).$$

Note que cada fila de la tabla produce una actualización de $\boldsymbol{x}$, que serían los parámetros a minimizar del un modelo. Entonces cuando terminemos de iterar en el conjunto A habremos actualizado n veces los parámetros. Por tanto una sola pasada en la tabla de datos representa n actualizaciones en los parámetros mientras que con Descenso de Gradiente una sola pasada sobre los datos representa una sola actualización.

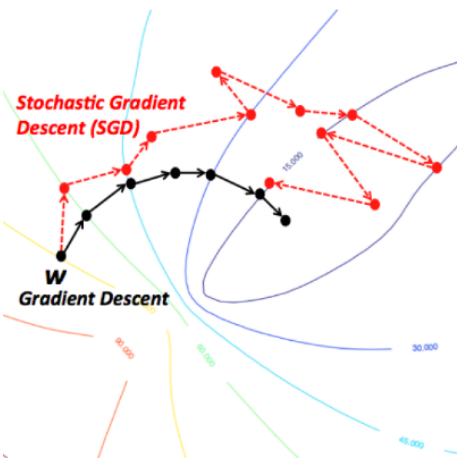
El algoritmo para el planteamiento generalizado del Descenso de Gradiente Estocástico (Mini Batch SGD) es el siguiente.

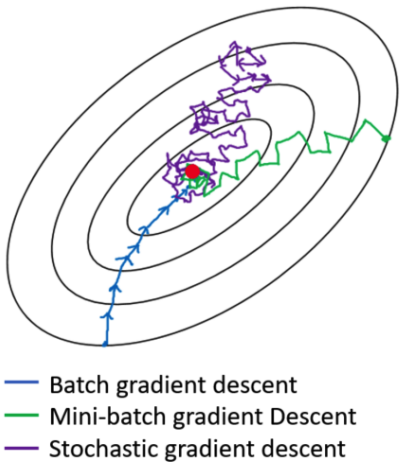
1. Seleccione una learning rate $\alpha > 0$ y parámetros iniciales $\boldsymbol{x}_0$.
2. Repetir hasta obtener un mínimo aproximado (epochs).
   a) Considere una permutación aleatoria A del conjunto $\{1, \ldots, n\}$
   b) Divida A en baches disjuntos del mismo tamaño $\mathcal{C} = \{\mathcal{B}_i\}$ para $i = 1, \ldots, \lceil \frac{n}{|\mathcal{B}|} \rceil$.
   c) De forma secuencial para cada $\mathcal{B}_i$ en $\mathcal{C}$ se actualiza $\boldsymbol{x}$ de la siguiente forma (iteraciones)
   $$\boldsymbol{x} := \boldsymbol{x} - \alpha \nabla f_{\mathcal{B}_i}(\boldsymbol{x}).$$

Nota. No siempre se puede dividir A en baches disjuntos del mismo tamaño, en ese caso el último batch tendría un tamaño reducido.

Comparación Gráfica

El objetivo de esta sección es comparar de forma gráfica los métodos de Descenso de Grandiente y Descenso de Gradiente Estocástico y ver cómo la definición de cada uno afecta la forma en que se mueven hacia el óptimo.





En la figura de la izquierda podemos ve como el movimiento de GD es mucho más eficiente para moverse en dirección al óptimo y lo hace de una forma más estable, mientras que el SGD tiene un movimiento más errático lo cual se debe a la naturaleza aleatoria del método. Note además que a diferencia del GD, el SGD no siempre avanza en dirección al óptimo, hay iteraciones en que aumenta el costo en lugar de disminuirlo, lo cual se debe a que usamos una aproximación del gradiente y no el gradiente exacto.

En la figura la derecha mostramos las 2 versiones del SGD estudiadas: La línea morada corresponde al SGD original, el cuál usa el bacth size de 1, y la línea verde corresponde al Mini Batch SGD con batch size entre 1 y el número de filas de la tabla.

Note al igual que antes que GD tiene el descenso más estable entre los 3, siempre avanzando en la dirección al óptimo. Por otro lado el SGD presenta la mayor variación en su movimiento, y el Mini Batch SGD es el caso intermedio. Claramente conforme aumentemos el batch size de 1 a n se espera que el descenso sea cada vez más estable y que el tiempo de ejecución se reduzca.

Convergencia del Descenso de Gradiente Estocástico

El Descenso de Gradiente Estocástico utiliza una aproximación del gradiente en cada iteración. Ahora la pregunta es: *¿cuál es el costo de utilizar un gradiente aproximado?* La

respuesta consiste en que la tasa de convergencia es más baja que con el algoritmo de Descenso de Gradiente. Veamos el siguiente teorema.

Teorema 1.2. *Sea $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ una función convexa L–Lipchitz y sea $x^* = \arg\min_x f(x)$. Considere un paso de SGD donde los estimadores $\boldsymbol{v}^{(k)}$ tienen varianza acotada: para todo $k \geq 0$, $\text{var}[\boldsymbol{v}^{(k)}] \leq \sigma^2$. Entonces para cualquier $k > 1$, SGD con paso de tamaño $t \leq 1/L$ satisface*

$$\mathbb{E}[f(\overline{x_k})] \leq f(x^*) + \frac{||x_0 - x^*||_2^2}{2tk} + \frac{\sigma^2 t}{2}$$

donde $\overline{x_k} = \dfrac{x_1 + \ldots + x_k}{k}$. En particular, para $k = \dfrac{\sigma^2 + L||x_0 - x^*||_2^2}{\epsilon^2}$ iteraciones basta con encontrar un valor óptimo x $2\epsilon-$ aproximada- en esperanza- haciendo $t = \dfrac{1}{\sqrt{k}}$.

Solución. El argumento es muy similar al caso determinístico.

$$\begin{aligned} f(x_{i+1}) &\leq f(x_i) + \langle \nabla f(x_i), x_{i+1} - x_i \rangle + \frac{L}{2}||x_{i+1} - x_i||_2^2 \\ &= f(x_i) - t||\nabla f(x_i)||_2^2 + \frac{Lt^2}{2}\left(||\nabla f(x_i)||_2^2 + \text{Var}(\boldsymbol{v}_t)\right) \end{aligned}$$

Entonces, tomando esperanza con respecto a la escogencia de $\boldsymbol{v}_t$, se tiene

$$\begin{aligned} \mathbb{E}[f(x_{i+1})] &\leq f(x_i) - t||\nabla f(x_i)||_2^2 + \frac{Lt^2}{2}\left(||\nabla f(x_i)||_2^2 + \text{Var}(\boldsymbol{v}_t)\right) \\ &\leq f(x_i) - t(1 - Lt/2)||\nabla f(x_i)||_2^2 + \frac{Lt^2}{2}\sigma^2 \\ &\leq f(x_i) - \frac{t}{2}||\nabla f(x_i)||_2^2 + \frac{Lt^2}{2}\sigma^2 \qquad \text{(Pues } Lt \leq 1\text{)} \end{aligned}$$

note que a diferencia del GD, el SGD no necesita ser monótona. Combinando las dos ecuaciones anteriores se tiene

$$\mathbb{E}[f(x_{i+1})] \leq f(x^*) + \langle \nabla f(x_i),\ x_i - x^* \rangle - \frac{t}{2}||\nabla f(x_i)||_2^2 + \frac{t}{2}\sigma^2$$

Ahora se hace una sustitución hacia atrás de los $\boldsymbol{v}_i$ en la ecuación $\mathbb{E}[\boldsymbol{v}_i] = \nabla f(x_i)$ y $||\nabla f(x_i)||_2^2 = \mathbb{E}[||\boldsymbol{v}_i||_2^2] - Var(\boldsymbol{v}_i) \leq \mathbb{E}[||\boldsymbol{v}_i||_2^2] - \sigma^2$:

$$\begin{aligned} \mathbb{E}[f(x_{i+1})] &\leq f(x^*) + \langle \mathbb{E}[\boldsymbol{v}_i],\ x_i - x^* \rangle - \frac{t}{2}\mathbb{E}[||\boldsymbol{v}_i||_2^2] + \frac{t}{2}\sigma^2 \\ &= f(x^*) + \mathbb{E}\left[\langle \boldsymbol{v}_i,\ x_i - x^* \rangle - \frac{t}{2}||\boldsymbol{v}_i||_2^2\right] + t\sigma^2 \end{aligned}$$

Procediendo como en el caso de GD, se completa cuadrado y se llega al siguiente resultado

$$\begin{aligned}\mathbb{E}[f(x_{i+1})] &\leq f(x^*) + \mathbb{E}\left[\frac{1}{2t}\left(||x_i - x^*||_2^2 - ||x_i - x^* - t\boldsymbol{v}_i||_2^2\right)\right] + t\sigma^2 \\ &= f(x^*) + \mathbb{E}\left[\frac{1}{2t}\left(||x_i - x^*||_2^2 - ||x_{i+1} - x^*||_2^2\right)\right] + t\sigma^2\end{aligned}$$

Sumando sobre todas las ecuaciones, se tiene

$$\sum_{i=0}^{k-1}(\mathbb{E}[f(x_{i+1})] - f(x^*)) \leq \frac{1}{2t}\left(||x_0 - x^*||_2^2 - \mathbb{E}\left[||x_k - x^*||_2^2\right] + kt\sigma^2\right) \leq \frac{||x_0 - x^*||_2^2}{2t} + kt\sigma^2$$

Finalmente, al ser f una función convexa se cumple

$$\sum_{i=0}^{k-1}(\mathbb{E}[f(x_{i+1})] - f(x^*)) = \mathbb{E}\left[f(x_1) + \cdots + f(x_k)\right] - kf(x^*) \geq k\mathbb{E}[f(\overline{x}_k)] - kf(x^*)$$

Combinando las dos ecuaciones, se obtiene el resultado deseado. □

1.4. Modelación y Optimización

Introducción

Un modelo es una representación matemática de un proceso o fenómeno con el cual se pueden realizar predicciones. Un modelo paramétrico es entonces una representación en la que un grupo finito de parámetros contiene toda la información necesaria para realizar dichas predicciones.

Algunos ejemplos de modelos paramétricos más usados en Machine Learning son: Regresión Lineal, Regresión Logística, Máquinas de Soporte Vectorial. Entre los beneficios de usar estos modelos paramétricos está:

- ⋆ Facildiad de entender el método e interpretar sus resultados.
- ⋆ La velocidad de ajuste a los datos es muy buena.
- ⋆ No requieren una cantidad muy grande de observaciones para entrenarse.

Mientras que entre sus desventajas está que al elegir funciones de predicción que obedecen representaciones analíticas, se tiene restricción que se logren encontrar valores a los parámetros que ajusten bien a los datos. Esto se conoce como el *trade off* entre sesgo y varianza.

Encontrar los parámetros de este tipo de modelos hace referencia a resolver un problema de optimización para minimizar el *error* cometido entre las predicciones de un modelo, y los valores reales de estas predicciones vistas en nuestro conjunto de entrenamiento (un subconjunto de los datos iniciales, que se usa para entrenar el modelo mediante una rutina de optimización, mientras que los datos restantes se usan para evaluarlo).

Supongamos que tenemos N observaciones de puntos $\boldsymbol{x}_i$ que pertenecen a dos clases indicadas por -1 y 1. Esos puntos vienen en pares con sus etiquetas, las cuales vamos a denotar y_i, entonces nuestros datos son de la forma $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$. Nuestro objetivo será entrenar un modelo capaz de predecir la etiqueta a cada punto, por lo cual nos encontramos en el contexto de un problema de *clasificación binaria*.

Una función de pérdida $J_\Theta(\boldsymbol{x}, \boldsymbol{y})$ es una función que permite evaluar que tan bien se ajusta un modelo a las observaciones para una selección de parámetros Θ, y es esta la que se busca minimizar al considerar Θ variable, pues se toma no negativa.

Modelos a Considerar

Nuestra estrategia es implementar primeramente el modelo de máquina de soporte vectorial (SVM) junto con la función de pérdida *Hinge Loss* ya que esta es una combinación muy usual en modelos de clasificación de dos categorias. El objetivo de las SVM es encontrar un hiperplano en el espacio N - dimensional que separe a los datos de manera tal que las dos etiquetas presentes queden en regiones distintas delimitadas por el mismo, y a su vez que se genere el mayor margen posible a ambas categorías.

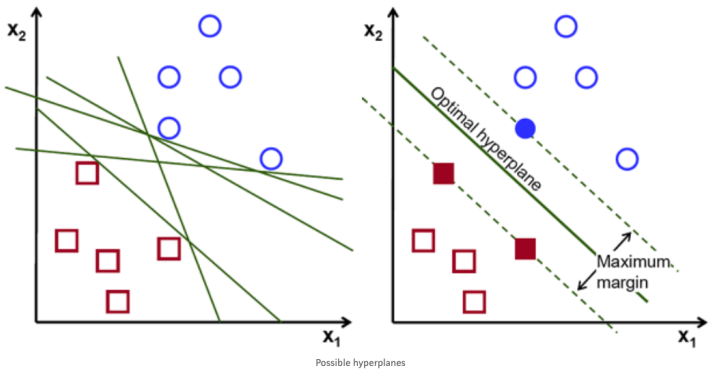


Figura 1.2: Interpretación geométrica de SVM

Para predecir el signo de la etiqueta y asociada a una observación $\boldsymbol{x}$ se usa

$$\text{signo}\left(\boldsymbol{w}^t \cdot \boldsymbol{x} + b\right)$$

donde $\boldsymbol{w}$, b son parámetros por optimizar del hiperplano dado por $\boldsymbol{a} \cdot \boldsymbol{w} + b = 0$.

Mientras que la *Hinge Loss* penaliza el error de predicción asociado a un elemento $(\boldsymbol{x}_i, y_i)$ mediante $(1 - y_i(\boldsymbol{w}^t \cdot x_i + b))_+$, y así la función de costo a minimizar es

$$\frac{1}{N}\sum_{i}^{N}(1 - y_i(\boldsymbol{w}^t \cdot x_i + b))_+ + \frac{\lambda||\boldsymbol{w}||_2^2}{2}$$

al considerarse función de pérdida con regularización, donde λ es un hiperparámetro de la rutina de optimización.

Algunas características de esta función de pérdida son

- ⋆ No solo penaliza predicciones incorrectas, también predicciones correctas con baja probabilidad.
- ⋆ Es una función convexa.
- ⋆ No es una función diferenciable, pero tiene subgradiente con respecto a pesos $\boldsymbol{w}$, a. A la hora de realizar la optimización, se calcula el gradiente por partes. Posible justificación para esto es que está función no es diferenciable en $\{1 = y_i(\boldsymbol{w}^t \cdot x_i + b)\}$, sin embargo resulta poco probable que realmente se vaya a alcanzar estos puntos por temas de punto flotante.

Observación. *En el caso práctico que se va a presentar se toma la Hinge Loss cuadrática, que castiga la predicción asociada a una observación* $\boldsymbol{x_i}$ *mediante* $\left(1 - y_i(\boldsymbol{w}^t \cdot \boldsymbol{x_i} + b)\right)_+^2$. *Este es un suavizamiento usado ya que la función de costo se vuelve ahora diferenciable. Adicionalmente, se interpreta como función de costo que castiga más fuertmenete a observaciones que violen el margen.*

Observación. *La teoría de subgradientes y subgradiete descendente vienen a tratar este tipo de problemas.*

También bajo la misma notación anterior pero considerando $y_i \in \{0, 1\}$ consideramos el modelo de Regresión Logística bajo la función de pérdida *Log Loss*. Un modelo de Regresión Logística tiene la siguiente ecuación:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \boldsymbol{x}^t \cdot \boldsymbol{\beta})}}$$

Donde $\beta_0, \boldsymbol{\beta}$ son los coeficientes del modelo por optimizar. Este modelo genera valores entre 0 y 1, donde $p(\boldsymbol{x_i}) \in [0, 1]$ se interpreta como la probabilidad de predecir $y_i = 1$. Mientras que la *Log Loss* penaliza el error de predicción de una observación $\boldsymbol{x_i}$ mediante $y_i \log(p(y_i)) + (1 - y_i) log(1 - p(y_i))$, y consecuentemente la función de costo a optimizar para ajuste del model está dada por

$$\frac{-1}{N} \sum_i^N y_i \log(p(y_i)) + (1 - y_i) log(1 - p(y_i))$$

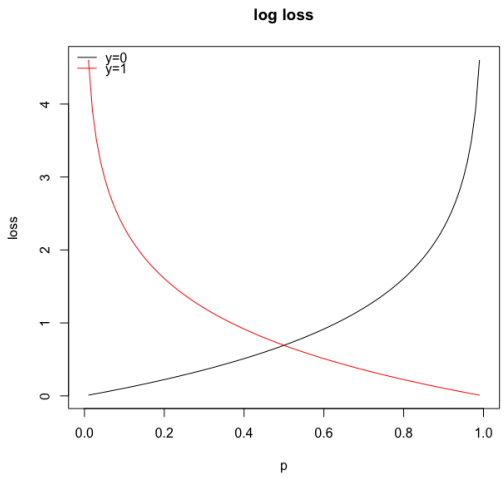


Figura 1.3: Función de pérdida vista como función a trozos

Algunas características de está función de pérdida son:

- ⋆ La pérdida asociada a una observación incrementa de manera logarítmica conforme la probabilidad asociada de predicción se aleja de predecir etiqueta real.
- ⋆ Es diferenciable.

1.5. Aplicación a un Problema de Modelado

Descripción de los Datos

Para la implementación de un modelo de clasificación y analizar el desempeño de las dos rutinas optimización de interés contamos con una tabla de datos de tipo financiero, la cual está formada por 10 000 filas y 13 columnas. Seguidamente se explican las variables que conforman la tabla.

- ⋆ `IdCliente`: Número de identificación del cliente en el banco.
- ⋆ `Apellido`: Apellido del cliente.
- ⋆ `ScoringCrediticio`: Resume el record crediticio del cliente.
- ⋆ `Pais`: País donde reside el cliente.
- ⋆ `Genero`: Género del cliente.
- ⋆ `Edad`: Edad del cliente.
- ⋆ `IdCuenta`: Número de identificación de la cuenta principal del cliente.
- ⋆ `BalanceCuenta`: Balance total en las cuentas del cliente.
- ⋆ `CantidadProductos`: Cantidad del productos del cliente con el banco.
- ⋆ `TarjetaCredito`: Indica si el cliente tiene tarjeta de crédito con el banco o no.
- ⋆ `Activo`: Indica si el cliente ha estado activo en los últimos meses o no (1 = Sí, 0 = No).
- ⋆ `SalarioEstimado`: Salario estimado del cliente.
- ⋆ `DejaBanco`: Indica si el cliente deja sus negocios con el banco o no (1 = Sí, −1 = No).

La tabla hace referencia a un problema de *churning* que se refiere a poder identificar si un cliente va a dejar sus negocios con banco o no, representado en la columna `DejaBanco`. Este problema es de relevancia para las instituciones ya que es menos costoso mantener los clientes existentes que adquirir nuevos clientes. Además, la salida de clientes limita el crecimiento de las instituciones y por esto las empresas deberían tener métodos para monitorear este tipo de salidas de clientes y encontrar oportunidades de mejora.

Note que esta tabla de datos representa un problema desbalanceado en el sentido de que la proporción de clientes que sí dejan sus negocios con el banco es mucho mejor que la proporción que no, 20.37 % y 79.63 % respectivamente.

Tratamiento de los Datos para Modelación

Se descartan las variables `Apellido, IdCliente, IdCuenta` por ser identificadores únicos de cada cliente, la variable categórica `Pais` se transforma en variables *dummy*, adicionalmente, la variable `Genero` se codifica como 1 para Masculino y 0 para femenino y de manera análoga se hace para variable `TarjetaCredito` y `Activo`. Luego de dichas transformaciones, se procede a centrar (restando la media) y reducir (dividiendo entre la desviación estándar) todas las variables pues esto mejora la convergencia de los métodos de descenso de gradiente lo cuáles no son invariantes a las escalas de las variables.

Aplicación de las Rutinas GD, SGD y sus Variantes

El objetivo de esta sección es aplicar las rutinas de Descenso de Gradiente y Descenso de Gradiente Estocástico, junto con algunas de sus variables, al problema de churning propuesto y realizar una serie de pruebas para comparar el comportamiento y desempeño de los optimizadores, además de poder seleccionar el mejor método para efectos de predicción.

La implementación se realiza con ayuda de la librería de *Matlab SGDLibrary* de PhD Hiroyuki Kasai, la cual está disponible en GitHub y fue la única librería gratuita que encontramos para la aplicación de estas rutinas de optimización.

Debido a que cada algoritmo empleado usa diferentes parámetros y necesita diferente número de evaluaciones en cada epoch o iteración es que en las pruebas se compara el costo obtenido con el número de evaluaciones de gradiente que requiere el algoritmo.

Para efectos de comparación en cada prueba usa los mismos parámetros (a menos que se indique lo contrario) según corresponda: 50 iteraciones GD, 50 epochs SGD, learning rate de 0.1, decreciente en cada iteración, y el mismo vector de parámetros iniciales. Las pruebas consisten en hacer variar algunos de estos y otros parámetros para estudiar el comportamiento de las rutinas. Las pruebas para comprar el costo se realizan con el enfoque

train y test, con 90 % de la tabla para train; las peubas para seleccionar el mejor modelo se realizan con Cross Validation, y para ambas se usa el método SVM.

- ⋆ **Prueba 1**. Vamos a comprar los algoritmos de *line search* para GD disponibles en la librería para ver el desempeño de cada uno. Los algoritmos considerados son *backtracking*, *strong wolfe* y *tfocs backtracking* los cuáles detallamos un poco a continuación.

  (a) Backtracking: Este algoritmo empieza con step size grande e iterativamente lo va decreciendo hasta que se observe un descrecimiento de la función objetivo que corresponda al decrecimiento esperado basado en el gradiente de la función.

  (b) Strong Wolfe: Este algoritmo impone una serie de restricciones en el step size, las cuales buscan garantizar la convergencia del gradiente a 0.

  (c) Tfocs Backtracking: El cual es una modificación del algoritmo Backtracking usando duales y suavizamientos.

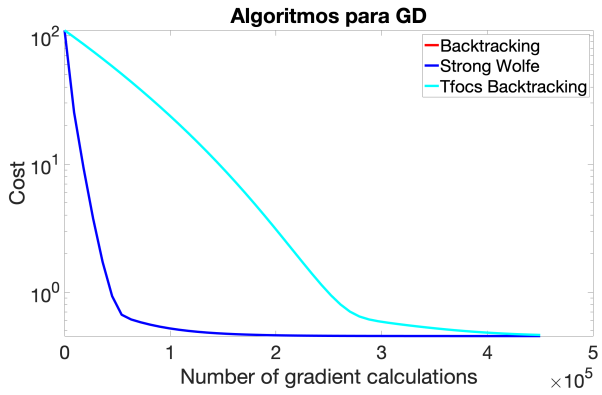


Figura 1.4: Prueba 1

Aunque no es apreciable en el gráfico, solo el algoritmo tfocs backtracking da diferente a los demás, y es el que requiere casi el doble de evaluaciones en el gradiente para estabilizar el costo. Los otros dos hacen un buen trabajo minimizando y estabilizando el costo.

En lo que sigue vamos utilizar el algoritmo por defecto y recomendado para GD, que es el backtracking.

- ⋆ **Prueba 2.** Esta prueba consiste en comparar SGD usando diferentes selecciones para el batch size de 1, 32, 64, 128 y $n = 9\,000$ (número de filas de la tabla de entrenamiento). Los primeros cuatro son los batch sizes más recomendados en la práctica.

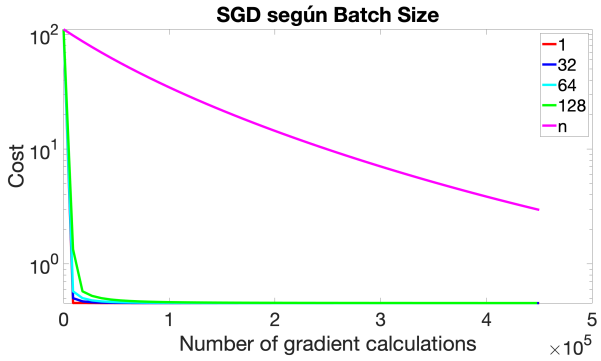


Figura 1.5: Prueba 2

Podemos ver que el batch size 1 es que el más rápido minimiza y estabiliza el costo, seguido de 32, 64, 128 y n, el cual no es capaz de minimizar el costo al nivel de los otros.

Observamos es que conforme aumenta el bacth size el algoritmo usa mayor cantidad de evaluaciones de gradiente para estabilizar el costo. Lo óptimo sería entonces usar batch size de 1. Sin embargo conforme el batch size crece, decrece el tiempo de ejecución. Por tanto en la practica no siempre es posible usar 1 y se tiende a usar 32 o 64. Para nuestra tabla un batch size 1 no implica un extenso tiempo de ejecución por lo que este es el que usaremos para las demás pruebas.

★ **Prueba 3**. En esta prueba vamos a probar el SGD usando distintas opciones para el decrecimiento de la learning rate. Tenemos 4 opciones de decrecimiento.

(a) Decremiento 0: En la iteración i usa la learning rate original lr_0.

(b) Decremiento 1: En la iteración i usa la learning rate $\frac{lr_0}{1+lr_0\,\lambda\,i}$, donde λ representa una proporción de lr_0, por defecto usamos $\lambda = 0{,}1$. Este ecrecimiento 1 es el más comúnmente usado.

(c) Decremiento 2: En la iteración i usa la learning rate $\frac{lr_0}{1+i}$.

(d) Decremiento 3: En la iteración i usa la learning rate $\frac{lr_0}{\lambda+i}$.

El siguiente gráfico muestra los 4 tipos de decrecimiento con $lr_0 = 0{,}1$ y $\lambda = 0{,}1$.

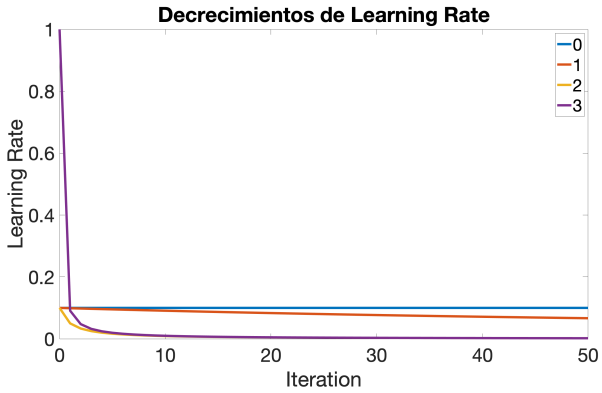


Figura 1.6: Decrecimientos 0, 1, 2 y 3

Veamos cómo se desempeñan las rutinas de optimización.

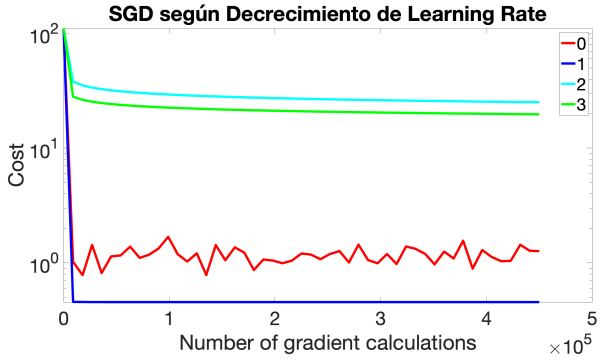


Figura 1.7: Prueba 3

Como vemos cuando se usa una learning rate constante el método SGD no es capaz de estabilizar el costo, esto se debe a que al tener pasos constantes el método brinca alrededor de un óptimo sin poder acercarse lo suficiente.

Cuando usamos una learning rate decreciente los métodos son capaces de estabilizar el costo, sin embargo vemos que los decrecimientos 2 y 3 son incapaces de disminuir el costo lo suficiente y esto se debe a que en pocas iteraciones la learning rate es demasiado pequeña, lo cual le impide al algoritmo avanzar lo suficiente y posiblemente queda atrapado en mínimos locales de mala calidad. Notamos que el método de decrecimiento 1 es el más eficiente ya que logra estabilizar el costo al mínimo entre los 4 métodos, esto se debe a que el decrecimiento de la learning rate es menos drástico que el usado en 2 y 3, lo cual le permite avanzar un poco más rápido hasta

el óptimo pero sin brincárselo como en el caso fijo. Este método de decrecimiento de será usando de aquí en adelante para todos los métodos derivados de SGD.

★ **Prueba 4.** Esta prueba consiste en usar SGD para acercar los parámetros lo más posible al óptimo y luego terminar de acercarnos aún más con GD. Se compara el resultado con la aplicación de GD desde el mismo punto inicial que se usó en SGD, para ver si hay mejora en los resultados al aplicar primero SGD.

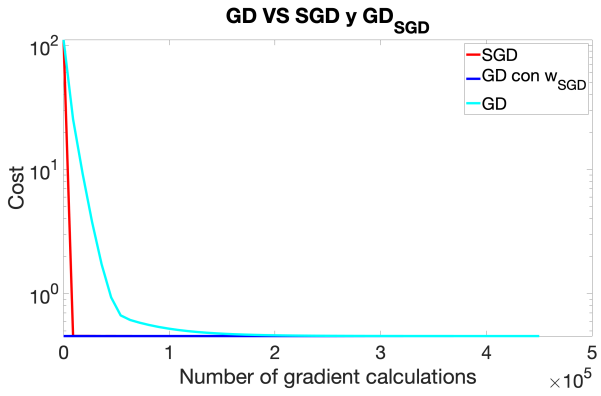


Figura 1.8: Con decrecimiento 1

SGD con decrecimiento 1 realiza un buen trabajo disminuyendo el costo. En este caso la aplicación previa de SGD nos acerca mucho al óptimo por lo que al aplicar GD, partiendo del punto en que nos deja SGD, casi no observamos cambios en el costo. Al aplicar GD sin usar SGD previamente vemos que el método alcanza el mismo costo, sin embargo emplea muchas más evaluaciones del gradiente. Note que si hubiéramos usado 1×10^5 evaluaciones de gradiente el primer enfoque utilizado hubiera estabilizado el costo pero el segundo no, lo cual representa un punto a favor del enfoque 1.

Veamos el comportamiento de esta prueba pero usando SGD con la learning rate fija y con decrecimiento 3. Vemos que SGD disminuye el costo, sin embargo no lo estabiliza en el caso de learning rate fija, y no lo minimiza lo suficiente en ambos casos. Al aplicar GD partiendo del punto en que nos deja SGD se tiene que el costo disminuye aún más, y se estabiliza mucho más rápido que al aplicar GD sin usar SGD previamente, lo cuál nuevamente representa una mejora respecto solo a usar GD o SGD.

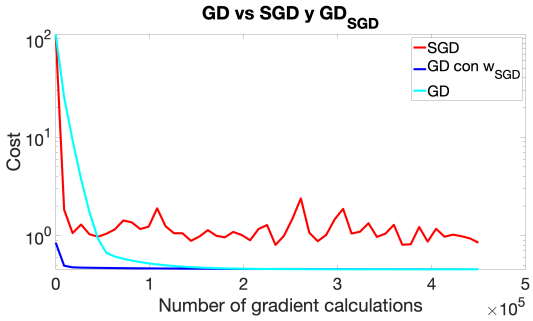


Figura 1.9: Con learning rate fija

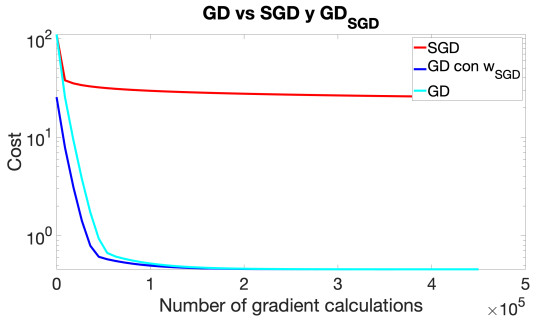


Figura 1.10: Con decrecimiento 3

⋆ **Prueba 5.** Seguidamente ponemos a competir los métodos GD, SGD y algunas de sus variantes para ver su capacidad para disminuir y estabilizar el costo. Las variantes del SGD empleadas son SDG CM, Adam y SVRG de los cuáles detallamos un poco a continuación.

(a) SGD CM (SGD with classical momentum): Este método está diseñado para acelerar la convergencia del SGD recorriendo más eficientemente las *curvaturas patológicas* de la superficie de costo. En lugar de usar solo el gradiente del paso actual para guiar el descenso, este método también acumula y da un peso a los gradientes de pasos anteriores para ayudar a determinar la dirección del descenso. Los gradientes anteriores más recientes tienen un mayor peso.

(b) Adam: Fue especialmente diseñado para entrenar redes neuronales. Usa diferentes learning rates para cada parámetro y emplea estimaciones del primer y segundo momento del gradiente para adaptar las learning rates, incorpora también el momentum al igual que SGD CM. En algunos casos puede encontrar peores soluciones que SGD.

(c) SVRG: Reduce la varianza del gradiente producto del muestreo aleatorio empleado, pues el gradiente estocástico es una aproximación aleatoria del gradiente total. El reducir la variancia permite que el modelo sea menos sensible al learning rate y su decrecimiento. Además no requiere guardar un histórico del gradiente, pero el número de evaluaciones de gradiente por iteración aumenta.

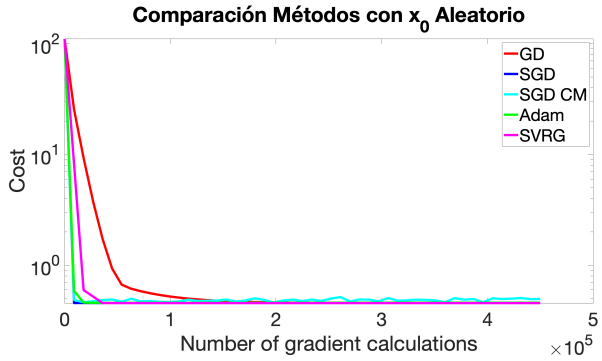


Figura 1.11: Prueba 5

Vemos de la figura anterior que en general todos los métodos hacen un buen trabajo disminuyendo el costo, sin embargo SGD parece ser el que lo hace en menos evaluaciones del gradiente, en segundo lugar está SGD CM y Adam, sin embargo SGD CM no lo disminuye tanto y no llega a ser tan estable como los otros métodos. Estas variaciones de SGD CM se pueden explicar por la incorporación del momentum, el cual se usa alcanzar convergencia de forma más rápida, pero dado que para este problema la convergencia ya es eficiente, la incorporación del momentum puede estar impidiendo que el método estabilice el costo. Por otro lado vemos como GD es el que toma mayor cantidad de evaluaciones del gradiente para estabilizar el costo, esta observación concuerda con la teoría, pues sabemos que el método SGD (y variaciones) se mueven más rápido hacia el óptimo que el método GD.

⋆ **Prueba 6.** Nuevamente ponemos a competir los métodos GD, SGD, SDG CM, Adam y SVRG para ver su capacidad para disminuir y estabilizar el costo, pero esta vez cambiando los parámetros de inicio. En pruebas anteriores hemos visto que los parámetros óptimos retornados son números pequeños alrededor de 0, por tanto tomamos como vector de parámetros iniciales un vector constante con el número 300.

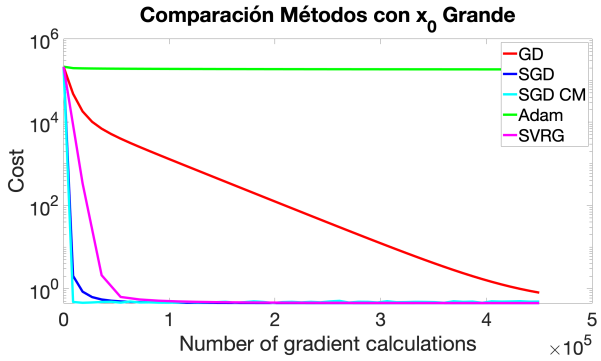


Figura 1.12: Prueba 6

En el gráfico notamos que ahora el comportamiento de los métodos es bastante diferente. Para nuestra sorpresa Adam ahora se desempeña pobremente, lo cual corresponde a lo mencionado en la definición del método cuando dijimos que en algunos casos puede desempeñarse mal. Por otro lado, ahora SGD CM es el más eficiente para disminuir el costo seguido de SGD y SVRG, los cuales al final presentan un costo muy parecido. Observamos además que GD es al que le toman más evaluaciones de gradiente el decrecer el costo (excepto por Adam) y no es capaz de decrecerlo al nivel de los otros métodos. Por último hacemos notar que SGD CM sigue presentado ligeras variaciones y que SGD y SVRG parecen ser los que alcanzan un mejor costo.

★ **Prueba 7. Selección del Modelo Final**. Por último queremos seleccionar el mejor modelo para efectos de predicción. Para esto vamos a comprar los métodos GD, SGD, SDG CM, Adam y SVRG usando los parámetros recomendados según las pruebas anteriores. Para comparar entre los métodos usamos *Cross Validation*, que el *gold standard* para comparación de métodos predictivos, usando $K = 5$ folds.

A continuación mostramos la tabla resumen que muestra las precisiones globales, las precisiones por categoría, y el área bajo la curva ROC (AUC) que se obtienen de la Cross Validation y usando el modelo SVM.

Cuadro 1: Cross Validation: Valores Promedio

Método	Precisión Global	Precisión −1	Precisión 1	AUC
GD	0.6460	0.6097	0.7879	0.7677
SGD	0.6436	0.6069	0.7869	0.7671
SGD CM	0.6057	0.5757	0.7231	0.7127
Adam	0.6511	0.6148	0.7928	0.7680
SVRG	0.6468	0.6096	0.7924	0.7680

Cuadro 2: Cross Validation: Desviación Estándar

Método	Precisión Global	Precisión −1	Precisión 1	AUC
GD	0.0116	0.0143	0.0274	0.0192
SGD	0.0089	0.0113	0.0142	0.0040
SGD CM	0.0292	0.0224	0.0581	0.0540
Adam	0.0094	0.0131	0.0232	0.0146
SVRG	0.0065	0.0040	0.0196	0.0094

Vemos los modelos desempeñan de formas muy similares, sin embargo de la Tabla 1 vemos que Adam es el que tiene mejores precisiones por categoría, y el que tiene mejor AUC, seguido del método SVRG. Por otro lado vemos que SGD CM es el que peores resultados tiene tanto en precisiones globales como en AUC.

Note que SG y SGD se desempeñan de forma muy similar en la tabla de datos, siendo GD ligeramente mejor.

Al ser éste un problema desbalanceado la precisión global no es una medida tan fiable, sin embargo se muestra como referencia.

En conclusión, para efectos de predicción el algoritmo seleccionado sería el Adam por brindar ligeramente mejores índices de calidad que el resto. Adam ofrece una precisión en la categoría −1 del 61.48 %, es decir que en promedio el modelo es capaz de detectar correctamente un 60.69 % de los clientes que no dejan el banco, y tiene una precisión en la categoría 1 del 79.28 % que significa que es capaz de predecir correctamente 79.28 % de los clientes que sí dejan el banco; además tiene un AUC de 0.7680 lo cual nos dice que si sacamos de forma aleatoria 2 clientes, el primero de clase −1 y el segundo de clase 1, entonces con probabilidad de 0.7680 el modelo le asigna una probabilidad mayor de ser churn al segundo. Es importante

hacer notar que según los resultados anteriores el modelo es mejor detectado los clientes que en efecto se van a ir del banco que los que no, lo cual puede ser algo bueno porque los clientes que en efecto se van del banco son los preocupantes.

La Tabla 2 nos muestra que los resultados de la Tabla 1 tuvieron poca variación a entre los folds, es decir que nuestros índices promedio son representativos.

⋆ **Prueba 8. Usando Regresión Logística.** Repetimos la prueba anterior pero usando un modelo de Regresión Logística con el fin de ver si se pueden mejorar los resultados obtenidos con Máquinas de Soporte Vectorial. No se muestran los resultados de las pruebas 1 a 7 pues los resultados son muy similares a los ya visto para Máquinas de Soporte Vectorial.

Cuadro 3: Cross Validation: Valores promedio

Método	Precisión Global	Precisión −1	Precisión 1	AUC
GD	0.6451	0.6094	0.7845	0.7669
SGD	0.6434	0.6082	0.7810	0.7663
SGD CM	0.6272	0.5932	0.7599	0.7514
Adam	0.6496	0.6140	0.7889	0.7671
SVRG	0.6441	0.6082	0.7845	0.7672

Note que los resultados mantienen las mismas relaciones que con el modelo de Máquinas de Soporte Vectorial, pero siendo solo un poco menores a las anteriores y de igual forma los resultados tuvieron poca variación entre los folds. Por tanto mantenemos nuestra elección de modelo final.

Mismo Problema con Diferente Modelo

Para efectos de comparar los resultados nuestros resultados con librerías estándar, se ajusta un modelo XGBoost en Python usando Sklearn y empleando Cross Validation para la comparación respectiva. Se emplea XGBoost pues tiende a dar muy buenos resultado en problemas de clasificación binaria. Los resultados son los siguientes. Mostramos también los resultados de Adam SVM de la prueba 7.

Cuadro 4: Cross Validation: Valores promedio

Método	Precisión Global	Precisión −1	Precisión 1	AUC
Adam	0.6511	0.6148	0.7928	0.7680
XGBoost	0.8510	0.9519	0.4690	0.8609

Note que XGBoost supera a Adam SVM en precisión global, precisión de la categoría −1 (no deja del banco) y en AUC. Sin embargo Adam SVM es superior en la precisión de la categoría 1 (deja el banco). La varianza obtenida de los resultados para XGBoost también es pequeña.

Por tanto la elección del modelo usar depende de los objetivos del negocio.

- ⋆ Si el objetivo es tener un modelo más preciso globalmente y con AUC alto, y queremos tener una forma eficiente de identificar los clientes que definitivamente no se van del banco entonces es mejor usar el XGBoost.
- ⋆ Si por otro lado queremos un modelo que sea capaz más bien de identificar mejor los clientes que sí van a dejar el banco sacrificando la precisión global y AUC, entonces lo recomendado es usar Adam SVM. Generalmente este es el objetivo.

1.6. Conclusiones

En este proyecto pudimos ver un recorrido por los métodos de optimización de Descenso de Gradiente y Descenso de Gradiente Estocástico y variantes, que son popularmente usados en técnicas de Deep Learning.

Vimos los detalles teóricos de cada método y las diferencias fundamentales de ambos, y la razón por la que SGD es una opción viable en casos en que GD no lo es.

Quizá no tuvimos la oportunidad de verdaderamente poner a prueba los métodos de SGD al usar en el proyecto una tabla de datos relativamente pequeña para los estándares actuales, pero sí pudimos estudiar el desempeño de las técnicas GD y SGD, y de las variaciones y mejoras que se han propuesto, además de poder observar cómo cada modificación de los parámetros puede afectar o mejorar el proceso de minimización.

Basados en la comparación con XGBoost el usar o no el modelo Adam SVM depende del objetivo de negocio planteado. Viendo que los resultados con técnicas SGD han sido relativamente positivas quizá es buena idea el explorar a futuro la aplicación de metodologías más específicas, como *sub-gradient descent* y *coordinate descent* que han sido especialmente diseñadas y que han demostrado ser buenas para el ajuste de modelos SVM.

Finalmente queremos señalar que nuestros resultados con Adam SVM no son para nada desalentadores, todo lo contrario. Obtenemos un modelo relativamente bueno detectando los clientes que sí dejan el banco, que es el objetivo generalmente buscado. Por otro lado, al evaluar los porcentajes de precisión se debe tener en mente que el problema de churning es desbalanceado, y tomando esto en consideración en realidad los índices de evaluación desempeñan mejor que el promedio.

1.7. Referencias

- ⋆ Hiroyuki, Kasai. (2018). SGDLibrary: A MATLAB library for stochastic optimization algorithms. Disponible en http://www.jmlr.org/papers/volume18/17-632/17-632.pdf
- ⋆ Hiroyuki, Kasai. (2018). SGDLibrary : Stochastic Optimization Algorithm Library in MATLAB. Disponible en https://github.com/hiroyuki-kasai/SGDLibrary
- ⋆ Hiroyuki, Kasai. (2018). GDLibrary : Gradient Descent Library in MATLAB. Disponible en https://github.com/hiroyuki-kasai/GDLibrary
- ⋆ Meka, Raghu. Notes on convergence of Gradient descend.
- ⋆ Nocedal, Jorge. and Wright, Stephen. (2006). Numerical Optimization. United States: Springer.
- ⋆ Tibshirani, Ryan. (2013). Gradient descend convergence. University of California, Berkeley.
- ⋆ Tibshirani Ryan, Hastie Trevor, Friedman Jerome (2008). The Elements of Statistical Learning, Springer, Second Edition.

$$y_{i,j} = \beta^T \mathrm{x_{i,j}} + \mathrm{u_i^T v_j} + a_i + b_j + \varepsilon_{i,j}$$
$$(\mathrm{u_1}, \mathrm{u_1}), \ldots, (\mathrm{u_n}, \mathrm{u_n}) = \mathrm{i.i.d} N_2\,(0, \Psi)\; a$$

1.8. Código Matlab

Seguidamente mostramos las funciones propias implementadas para el proyecto. Las demás funciones usadas fueron tomadas de las librerías SGDLibrary y GDLibrary de Matlab.

```
%%%Funci n propia para dividir la tabla en variables
   predictoras y variable a predecir
function [data_x, data_y] = DataDivisionXy(data, y_col)

% Data X
data_x = data;
data_x(:, y_col) = [];

% Data y
data_y = data(:, y_col);

end


%%%Funci n propia para divir la tabla en train test
function [data_training, data_testing] =
   DataDivisionTrainTest(data, p)

% Permutation of idexes
[n, ~] = size(data);
permutation_idx = randperm(n);

% Training testing split
training_idx = permutation_idx(1:round(p * n));
testing_idx = permutation_idx(round(p * n) + 1:end);

% Training set
data_training = data(training_idx, :);

% Testing set
data_testing = data(testing_idx, :);

end


%%%Funci n propia para hacer validaci n con train test
function [t_w, t_info, tt_accuracy, tt_class_accuracy, tt_auc
   ] = TrainTestValidation(data_train, data_test, y_col,
   problem_model, optimization_solver, solver_options,
```

```
    positive_class)
36
37 %Train and test X y division
38 [training_x, training_y] = DataDivisionXy(data_train, y_col);
39 [testing_x, testing_y] = DataDivisionXy(data_test, y_col);
40
41 %Define the problem
42 problem = problem_model(training_x.', training_y.', testing_x
    .', testing_y.');
43
44 %Run optimization solver
45 [t_w, t_info] = optimization_solver(problem, solver_options);
46
47 %Run prediction
48 [y_pred, y_scores] = problem.prediction(t_w);
49
50 %Evaluate results
51 [~, tt_accuracy, tt_class_accuracy, tt_auc] = Evaluate(
    problem, y_scores, y_pred, testing_y, positive_class);
52
53 end
54
55
56 %%%Funci n propia para hacer Cross Validation
57 function [cv_accuracy, cv_class_accuracy, cv_auc] =
    KFoldCrossValidation(data, y_col, K, problem_model,
    optimization_solver, solver_options, positive_class)
58
59 %Data X y division
60 [data_x, data_y] = DataDivisionXy(data, y_col);
61
62 %Create data partition with K folds
63 folds = cvpartition(data_y, 'KFold', K);  %Stratified K-fold
64
65 %Alocate vector results
66 acc = zeros(1, folds.NumTestSets);
67 class_acc = zeros(2, folds.NumTestSets);
68 auc = zeros(1, folds.NumTestSets);
69
```

```
70 %Recommends preallocating for speed
71 %acc = [];
72 %class_acc = [];
73 %auc = [];
74
75 for i = 1:folds.NumTestSets
76     %Training testing split
77     training_idx = folds.training(i);
78     testing_idx = folds.test(i);
79
80     %Training set
81     training_x = data_x(training_idx, :);
82     training_y = data_y(training_idx, :);
83
84     %Testing set
85     testing_x = data_x(testing_idx, :);
86     testing_y = data_y(testing_idx, :);
87
88     %Define the problem
89     problem = problem_model(training_x.', training_y.',
           testing_x.', testing_y.');
90
91     %Run optimization solver
92     w = optimization_solver(problem, solver_options);
93
94     %Run prediction
95     [y_pred, y_scores] = problem.prediction(w);
96
97     %Evaluate results
98     [~, acc_i, class_acc_i, auc_i] = Evaluate(problem,
           y_scores, y_pred, testing_y, positive_class);
99
100    %Update fold i results
101    acc(i) = acc_i;
102    class_acc(:, i) = class_acc_i;
103    auc(i) = auc_i;
104
105    %acc = [acc, acc_i];
106    %class_acc = [class_acc, class_acc_i];
```

```
107         %auc = [auc, auc_i];
108     end
109
110     %Average and Standar Deviation of folds results
111     cv_accuracy = [mean(acc), std(acc)];
112     cv_class_accuracy = [mean(class_acc, 2), std(class_acc, 0, 2)
            ];
113     cv_auc = [mean(auc), std(auc)];
114
115     end
```