Algoritmos de Boosting para Modelos de Clasificación y Regresión Lineal

Nicolás Rugeles Ospina

Universidad de los Andes
Facultad de Ciencias, Departamento de Matemáticas
Bogotá, Colombia
2022

Algoritmos de Boosting para Modelos de Clasificación y Regresión Lineal

Nicolás Rugeles Ospina

Trabajo de grado presentado como requisito parcial para optar al título de:
Matemático

Director:
Ph.D. Mauricio Junca

Universidad de los Andes
Facultad Ciencias, Departamento de Matemáticas
Bogotá, Colombia
2022

Resumen

El propósito de este trabajo es estudiar diferentes aplicaciones del método de Boosting en el contexto de aprendizaje de máquinas. Se abordan diferentes algoritmos de Boosting para crear modelos aditivos tales como Adaboost y LogitBoost; popularizados a finales de los años 90 por su buen rendimiento. Se prueba que estos algoritmos crean de forma iterativa un modelo de regresión logística aditivo optimizando localmente su función de costo correspondiente.

Asimismo, se presenta un caso general de Boosting dentro de un espacio funcional $(S, \langle , \rangle)$. En este caso, se estudian algoritmos que no dependan directamente de la función de costo asociada. Se define un funcional de costo y un diferencial de forma discreta sobre el espacio de funciones. Estos nos permiten usar el método de descenso de gradiente para definir algoritmos de Boosting.

Por otro lado, se estudian algoritmos de Boosting para resolver problemas de regresión lineal. Se estudian los algoritmos de LS-Boost(ε) y FS$_\varepsilon$ para resolver el problema de mínimos cuadrados:

$$\text{LS}: \quad L_n^* := \min_{\beta} L_n(\beta) := \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\beta\|_2^2$$
$$\text{s.t.} \quad \beta \in \mathbb{R}^p,$$

donde la matriz $X = [X_1, ..., X_p] \in \mathbb{R}^{n\times p}$ es la matriz de características y $\mathbf{y} \in \mathbb{R}^n$ es el vector de respuesta. Se muestra que cada uno de estos algoritmos corresponde a una instancia de descenso de subgradiente. Por consecuente, se derivan algunas propiedades de convergencia de los algoritmos. Por último, se estudia una aproximación al problema de mínimos cuadrados regularizado:

$$\text{LASSO: } L_{n,\delta}^* := \min_{\beta} \frac{1}{2n}||\mathbf{y} - \mathbf{X}\beta||_2^2$$
$$\text{s.a } ||\beta||_1 \leq \delta.$$

Índice general

Capítulo 1

Boosting en Clasificación

1.1. Primeros Algoritmos

El primer algoritmo de Boosting se desarrolló en 1990 por Robert Schapire. La idea de Schapire era mostrar que los algoritmos considerados como 'weak learners' podían mejorar considerablemente su desempeño entrenando más clasificadores sobre los datos ligeramente modificados. Se entiende de los 'weak learners' o clasificador débil como algoritmos de clasificación con un rendimiento bajo.

La primera idea de Boosting consistía en agrupar tres modelos con el fin de poder crear un voto mayoritario para clasificar problemas de dos clases. Para empezar, se entrena un modelo h_1 sobre N_1 datos tomados de una muestra. Posteriormente, se entrena un segundo modelo sobre una nueva muestra de N_2 datos de los cuales la mitad están mal clasificados por h_1. Por último, se entrena h_3 en una muestra de N_3 datos para los cuales h_1 y h_2 clasifican de formas distintas. Teniendo estos tres clasificadores, podemos crear h_{Boost} como el voto mayoritario de los tres anteriores. Intuitivamente, h_{Boost} es un mejor clasificador que cualquiera de los otros h_i, teniendo en cuenta que cada h_i tiene una eficacia mayor al 50 % y, al entrenarlos, la mitad

de la muestra proviene de los errores del anterior.

Schapire concluyó en 'Strength of Weak Learnability'(1990) que cualquier weak learner se puede usar para crear un algoritmo bastante eficiente y proporcionó varios métodos para convertirlos en algoritmos que consiguen alta precisión en sus predicciones. Con el fin de estudiar ejemplos específicos, esta sección estará centrada en los resultados expuestos en [FHT00].

1.1.1. AdaBoost Discreto y Real

Definición 1.1.1 (Modelos aditivos)**.** *Sean f_m modelos y c_m constantes definimos un modelo aditivo F como*

$$F(x) = \sum_{m=1}^{M} c_m f_m(x).$$

Unos de los ejemplos más avanzados y prácticos de algoritmos de Boosting son los de AdaBoost: el Discreto y el Real. El principio de ambos algoritmos es crear una familia de modelos. Cada modelo toma los errores de los anteriores y los prioriza para entrenarse. Posteriormente, se añade cada modelo a una suma ponderada que funciona para clasificar los datos de entrada. De esta forma, en cada iteración se construye un nuevo modelo que intenta predecir con mayor precisión los datos que el modelo anterior clasificaba de manera errónea.

Los algoritmos que vamos a presentar en esta sección permiten identificar datos dentro dos clases disjuntas. El conjunto de datos que vamos a denotar como S contiene N elementos y cada uno de estos lo podemos denotar como una pareja $(x, \mathbf{y})$ donde $x \in \mathbb{R}^n$ y $\mathbf{y} \in \{-1, 1\}$. Cabe aclarar que x es el vector de características; en cada componente puede tener un conjunto discreto de posibilidades y $\mathbf{y}$ representa la clase a la cual pertenece cada dato. Además, cada uno de los datos va a tener un peso asignado por una función de densidad que denotamos como $w(x, y)$ y que llamamos función de pesos.

Definición 1.1.2 (Esperanza condicional ponderada)**.** *Sea w una función de pesos, definimos la esperanza condicional ponderada de $h(x, \mathbf{y})$ como*

$$E_w[h(x, \mathbf{y})|x] = E[w(x, \mathbf{y})h(x, \mathbf{y})|x].$$

De la misma forma, podemos definir la esperanza ponderada como

$$E_w[h(x, \mathbf{y})] = E[w(x, \mathbf{y})h(x, \mathbf{y})].$$

Algoritmo 1 Algoritmo de AdaBoost Discreto

Inicializar: A cada uno de los datos se le asigna un peso $w_i = 1/N$ para $i = 1, ..., N$.

Para m desde 1 hasta M **hacer**

Entrenar un modelo $f_m(x) \in \{-1, 1\}$ teniendo en cuenta los pesos para cada uno de los datos.

Calcular el error ponderado $\text{err}_m = E_w[\mathbb{1}_{(\mathbf{y} \neq f(x))}] = \sum_{(x_i, \mathbf{y}_i)|f(x_i) \neq \mathbf{y}_i} w_i$.

Calcular el coeficiente de precisión $c_m = \ln((1 - \text{err}_m)/\text{err}_m)$.

Actualizar los pesos w_i para cada i, $w_i \leftarrow w_i \exp(c_m \mathbb{1}_{(\mathbf{y}_i \neq f(x_i))})$.

Renormalizar los pesos para que $\sum_i w_i = 1$.

fin Para

Retornar el clasificador dado por $\text{sgn}\left[\sum_{m=1}^{M} c_m f_m(x)\right]$.

Intuitivamente, el algoritmo de AdaBoost Discreto en cada iteración entrena un modelo y verifica cuánto es el error ponderado. Si este supera el 50 %, entonces con el coeficiente de precisión cambian las predicciones y se obtiene un estimador con un error menor al 50 %. Posteriormente, actualiza los pesos de tal forma que los datos que están bien clasificados van a tener menos influencia en la próxima iteración.

La mayor diferencia entre AdaBoost Real y AdaBoost Discreto consiste en la familia

de funciones a las que pertenecen los modelos f_m. En AdaBoost Real, para cada x tenemos que $f(x) \in \mathbb{R}$ y nos va a permitir dar una estimación de la probabilidad de que x pertenezca a la clase $\mathbf{y} = 1$ usando el modelo logístico.

Algoritmo 2 Algoritmo de AdaBoost Real

Inicializar: A cada uno de los datos se le asigna un peso $w_i = 1/N$ para $i = 1, ..., N$.

Para m desde 1 hasta M **hacer**

Entrenar un modelo $p_m(x) = \hat{P}_w(\mathbf{y} = 1|x) \in [0, 1]$ que estime la probabilidad de que x pertenezca a la clase $\mathbf{y} = 1$ teniendo en cuenta los pesos para los datos.

Definir $f_m(x) = \frac{1}{2} \ln \left(\frac{p_m(x)}{1-p_m(x)} \right)$.

Actualizar los pesos w_i para cada i, $w_i \leftarrow w_i \exp(-\mathbf{y}_i f_m(x_i)))$.

Renormalizar los pesos para que $\sum_i w_i = 1$.

fin Para

Retornar el clasificador dado por $\operatorname{sgn}\left[\sum_{m=1}^{M} f_m(x)\right]$

Al final del capítulo se van a analizar el desempeño de estos dos algoritmos con diferentes tipos de datos.

A continuación, veremos que los algoritmos de Adaboost Discreto y Real pueden interpretarse como procesos iterativos que adaptan una regresión logística aditiva y optimizan, a su vez, un criterio exponencial.

Definimos el criterio exponencial como $J(F) = E(e^{-\mathbf{y}F(x)})$ para cada modelo aditivo F resultante de una corrida de los algoritmos. Este criterio nos permite medir la eficiencia de cada modelo teniendo en cuenta que el objetivo es minimizar J.

Lema 1.1.3. *$E(e^{-\mathbf{y}F(x)})$ se minimiza en*

$$F(x) = \frac{1}{2} \ln \left(\frac{P(\mathbf{y} = 1|x)}{P(\mathbf{y} = -1|x)} \right).$$

Demostración. Sea $x \in \mathbb{R}^p$, queremos encontrar una función $F(x) \in \{G : \mathbb{R}^p \longrightarrow \mathbb{R}\}$

que minimice $E(e^{-\mathbf{y}F(x)})$. Tenemos que (usando el teorema de Tonelli)

$$E(e^{-\mathbf{y}F(x)}) = \sum_{\mathbf{y}\in\{-1,1\}} \int_{x\in\Omega} e^{-\mathbf{y}F(x)} g(x,\mathbf{y}) dx = \int_{x\in\Omega} \sum_{\mathbf{y}\in\{-1,1\}} e^{-\mathbf{y}F(x)} g(x,\mathbf{y}) dx,$$

donde $g(x,\mathbf{y})$ representa la función de densidad conjunta.

Si fijamos x, podemos minimizar $\sum_{\mathbf{y}\in\{-1,1\}} e^{-yF(x)} g(x,\mathbf{y})$ para cada x con el fin de minimizar la función objetivo $E(e^{-yF(x)})$. Además, podemos reescribir la esperanza

$$E(e^{-\mathbf{y}F(x)}|x) = \sum_{\mathbf{y}\in\{-1,1\}} e^{-\mathbf{y}F(x)} \frac{g(x,\mathbf{y})}{g_X(x)} = \frac{1}{g_X(x)} \sum_{\mathbf{y}\in\{-1,1\}} e^{-\mathbf{y}F(x)} g(x,\mathbf{y}),$$

y como x es fijo podemos tomar $1/g_X(x)$ como una constante y, por lo tanto, minimizar $\sum_{\mathbf{y}\in\{-1,1\}} e^{-\mathbf{y}F(x)} g(x,\mathbf{y})$ es equivalente a minimizar $E(e^{-\mathbf{y}F(x)}|x)$. Si ignoramos el término $1/g_X(x)$ nos queda,

$$\begin{aligned}
&E(e^{-\mathbf{y}F(x)}|x) = P(\mathbf{y}=1|x)e^{-F(x)} + P(\mathbf{y}=-1|x)e^{F(x)},\\
&\frac{\partial E(e^{-\mathbf{y}F(x)}|x)}{\partial F(x)} = -P(\mathbf{y}=1|x)e^{-F(x)} + P(\mathbf{y}=-1|x)e^{F(x)}.
\end{aligned}$$

Buscamos $F(x)$ tal que la derivada se anule, luego

$$\begin{aligned}
&-P(\mathbf{y}=1|x)e^{-F(x)} + P(\mathbf{y}=-1|x)e^{F(x)} = 0\\
\implies &-P(\mathbf{y}=1|x) + P(\mathbf{y}=-1|x)e^{2F(x)} = 0\\
\implies &P(\mathbf{y}=-1|x)e^{2F(x)} = P(\mathbf{y}=1|x)\\
\implies &e^{2F(x)} = \frac{P(\mathbf{y}=1|x)}{P(\mathbf{y}=-1|x)}\\
\implies &F(x) = \frac{1}{2}\ln\left(\frac{P(\mathbf{y}=1|x)}{P(\mathbf{y}=-1|x)}\right).
\end{aligned}$$

□

El anterior lema nos permite ver que el minimizador de $J(F) = e^{-\mathbf{y}F(x)}$ viene dado por la función inversa de la función logística (o sigmoide) dividida entre 2 al evaluarla en la probabilidad $p(x) = P(\mathbf{y} = 1|x)$. Equivalentemente podemos obtener un modelo logístico despejando la ecuación del lema. Nos queda que

$$P(\mathbf{y} = 1|x) = \frac{e^{2F(x)}}{1 + e^{2F(x)}}, \tag{1.1}$$

lo cual corresponde con el modelo logístico usual utilizado para modelar la probabilidad de que ocurra un evento.

Teorema 1.1.4. *Sea $\mathcal{F} := I^X$ el conjunto de funciones de $X \subset \mathbb{R}^n$ a I donde I es un subconjunto compacto de $\mathbb{R}$. Si tenemos que $f \in \mathcal{F}$ y $H : I \times X \to \mathbb{R}$ es una función continua sobre su primera coordenada tal que $H(f(x), x) > 0$ para todo $x \in X$ y $f \in \mathcal{F}$, entonces tenemos que*

$$E[H(f^*(x), x)] = E[\min_{g \in \mathcal{F}} H(g(x), x)],$$

lo que equivale a

$$E[H(f^*(x), x)] = E[\min_{z \in I} H(z, x)],$$

donde $f^* = \arg\min_{f \in \mathcal{F}} E[H(f(x), x)]$.

Demostración. Empecemos por reescribir $E[H(f^*(x), x)]$ usando la definición:

$$E[H(f^*(x), x)] = \int_X H(f^*(x), x) d\mu(x).$$

De la misma forma,

$$E[\min_g H(g(x), x)] = \int_X \min_g H(g(x), x) d\mu(x).$$

Tenemos que para cada x se cumple que $H(f^*(x), x) \geq \min_g H(g(x), x)$ entonces,

$$E[H(f^*(x), x)] \geq E[\min_g H(g(x), x)].$$

Para obtener la igualdad nos falta verificar que $E[H(f^*(x), x)] \leq E[\min_g H(g(x), x)]$. Supongamos que

$$E[H(f^*(x), x)] > E[\min_g H(g(x), x)],$$
$$\int_X H(f^*(x), x) d\mu(x) > \int_X \min_g H(g(x), x) d\mu(x).$$

Esto implica que existe un abierto $\hat{X} \subset X$ en el que

$$\int_{\hat{X}} H(f^*(x), x) d\mu(x) > \int_{\hat{X}} \min_g H(g(x), x) d\mu(x).$$

Definimos

$$h(x) = \begin{cases} \arg\min_{z \in I} H(z, x), & \text{si } x \in \hat{X}, \\ f^*(x) & \text{de lo contrario.} \end{cases}$$

tenemos que $h : X \to I$ está bien definida porque para cada x la función $H(a, x)$ es continua y $a \in I$ donde I es un conjunto compacto. Por el teorema de Weierstrass, tenemos que siempre se alcanza el mínimo de H.
Esto contradice el hecho que $f^* = \arg\min_{f \in \mathcal{F}} E[H(f(x), x)]$ y por lo tanto

$$E[H(f^*(x), x)] \leq E[\min_g H(g(x), x)].$$

De esta forma, podemos concluir que

$$E[H(f^*(x), x)] = E[\min_g H(g(x), x)].$$

□

Este teorema nos dice que si los valores a los que llegan las funciones clasificadoras son acotados y cerrados en $\mathbb{R}$ y la función objetivo es continua entonces encontrar la función que minimiza la esperanza de la función objetivo equivale a encontrar el valor de la función en cada punto minimizando la esperanza condicional. Es importante tener en cuenta que el conjunto de funciones admisibles que estamos tomando es extremadamente amplio y por esto llegamos a una igualdad.

Teorema 1.1.5. *El algoritmo de AdaBoost Discreto crea un modelo de regresión logística aditivo minimizando $J(F)$ en cada iteración.*

Demostración. Supongamos que tenemos un modelo dado por $F(x)$ y queremos mejorarlo usando una función $f(x)$ creando de esta forma un nuevo modelo $F(x) + cf(x)$ donde $c \in \mathbb{R}_{>0}$. Para mejorar el modelo queremos obtener un f^* que minimice $J(F + cf)$. Es decir, $f^* = \underset{f:X\to\{-1,1\}}{\arg\min}\ J(F + cf)$ y por lo tanto se debe cumplir que

$$\begin{aligned}
J(F + cf^*) &= \min_{f:X\to\{-1,1\}} J(F + cf) \\
&= \min_{f:X\to\{-1,1\}} E[e^{\mathbf{y}(F+cf)}] \\
&= \min_{f:X\to\{-1,1\}} \int_{x\in X} \sum_{\mathbf{y}\in\{-1,1\}} e^{-y(F+cf)(x)} g(x,\mathbf{y})dx \\
&= \min_{f:X\to\{-1,1\}} \int_{x\in X} \left(e^{(F+cf)(x)}\ P(\mathbf{y}=-1|x) + e^{-(F+cf)(x)}\ P(\mathbf{y}=1|x)\right) dx \\
&= \min_{f:X\to\{-1,1\}} \int_{x\in X} \left(e^{F(x)+cf(x)}\ P(\mathbf{y}=-1|x) + e^{-F(x)-cf(x)}\ P(\mathbf{y}=1|x)\right) dx.
\end{aligned}$$

Si tomamos $H(f(x), x) = e^{F(x)+cf(x)}\ P(\mathbf{y}=-1|x) + e^{-F(x)-cf(x)}\ P(\mathbf{y}=1|x)$ entonces podemos usar el teorema 1.1.4 para pasar de una minimización global de la función $J(F + cf)$ a una local para cada x.

$$J(F + cf^*) = \int_{x\in X} \left(\min_{z\in\{-1,1\}} e^{F(x)+cz}\ P(\mathbf{y}=-1|x) + e^{-F(x)-cz}\ P(\mathbf{y}=1|x)\right) dx,$$

Veamos para que valores de z se minimiza $e^{F(x)+cz}\ P(\mathbf{y}=-1|x)+e^{-F(x)-cz}\ P(\mathbf{y}=1|x)$

$$e^{F(x)}e^{cz}\ P(\mathbf{y}=-1|x)+e^{-F(x)}e^{-cz}\ P(\mathbf{y}=1|x)=\sum_{y\in\{-1,1\}} e^{-\mathbf{y}F(x)}e^{-ycz}\ P(y|x),$$

Si tomamos $w(x,\mathbf{y})=e^{-\mathbf{y}F(x)}$ entonces podemos reescribir lo anterior como

$$E_w[e^{-ycz}|x].$$

Teniendo en cuenta que la función exponencial es una función estrictamente creciente y $c>0$ es un valor fijo, se sigue que el z que minimiza la expresión es $\arg\min_{z\in\{-1,1\}} -E_w[yz|x]$. De igual forma, el z que minimiza la expresión es $\arg\max_{z\in\{-1,1\}} E_w[yz|x]$.

$$z=\begin{cases}1, & \text{si } E_w[y|x]=P_w(\mathbf{y}=1|x)-P_w(\mathbf{y}=-1|x)>0,\\ \text{-1}, & \text{de lo contrario.}\end{cases}$$

Con lo anterior, podemos definir explícitamente $f^*:X\to\{-1,1\}$

$$f^*(x)=\begin{cases}1, & \text{si } E_w[y|x]=P_w(\mathbf{y}=1|x)-P_w(\mathbf{y}=-1|x)>0,\\ \text{-1}, & \text{de lo contrario.}\end{cases}$$

Por último, veamos que $-E_w[\mathbf{y}F(x)]=E_w[y-f(x)]^2/2-1$

$$\begin{aligned}E_w[[y-f(x)]^2]/2-1 &= E_w[y^2-2\mathbf{y}F(x)+f(x)^2]/2-1\\ &= E_w[1-2\mathbf{y}F(x)+1]/2-1\\ &= 2E_w[-\mathbf{y}F(x)+1]/2-1\\ &= E_w[-\mathbf{y}F(x)]+1-1\\ &= -E_w[\mathbf{y}F(x)].\end{aligned}$$

Cabe notar que f^* maximiza $E_w[\mathbf{y}F(x)]$, es decir que minimiza $-E_w[\mathbf{y}F(x)]$. Por lo tanto, tenemos que la solución encontrada coincide con minimizar la diferencia

ponderada de cuadrados.
Lo encontrado anteriormente nos da una manera de determinar f^* de tal forma que se minimice la aproximación de J. Aún nos falta encontrar el valor de c^* y la actualización de los pesos asignados a cada dato que denotamos como $w(x, \mathbf{y})$. Tenemos que c^* es el valor que minimizar $J(F + cf)$. Para encontrarlo, primero definimos

$$\begin{aligned} \text{err} &= E_w[\mathbb{1}_{(y \neq f(x))}], \\ c^* &= \arg\min_c E_w[e^{-c\mathbf{y}f(x)}]. \end{aligned}$$

Ahora buscamos el c tal que la derivada de $E_w[e^{-c\mathbf{y}f(x)}]$ se anule.

$$\begin{aligned} &\frac{\partial E_w[e^{-c\mathbf{y}f(x)}]}{\partial c} = 0 \\ &\rightarrow -E_w[e^{-c\mathbf{y}f(x)}\mathbf{y}f(x)] = 0 \\ &\rightarrow e^{-c}(1 - \text{err}) - e^{c}\,\text{err} = 0 \\ &\rightarrow \frac{1 - \text{err}}{\text{err}} = e^{2c} \\ &\rightarrow c^* = \frac{1}{2}\ln\left(\frac{1 - \text{err}}{\text{err}}\right). \end{aligned}$$

Tenemos que c puede tomar valores negativos en el caso en el que el modelo tenga una precisión ponderada menor al 50 %. Ese caso, es equivalente a invertir todas las predicciones de $f(x)$ y tomar $c > 0$. Combinando la elección de f y c podemos definir la actualización de $F(x)$ en cada iteración,

$$F(x) \leftarrow F(x) + \frac{1}{2}\ln\left(\frac{1 - \text{err}}{\text{err}}\right) f^*(x)$$

.

La función $w(x, \mathbf{y}) = e^{-\mathbf{y}F(x)}$ también se actualiza de tal forma que siga siendo

coherente con la actualización de $F(x)$.

$$w(x, \mathbf{y}) \leftarrow w(x, \mathbf{y}) e^{-c^* f^*(x) y}.$$

Para poder encontrar la actualización descrita el algoritmo debemos usar que $-f(x)y = 2 \times \mathbb{1}_{(y \neq f(x))} - 1$

$$\begin{aligned} w(x, \mathbf{y}) e^{-c^* f^*(x) y} &= w(x, \mathbf{y}) e^{c^* (2 \times \mathbb{1}_{(y \neq f(x))} - 1)} \\ &= w(x, \mathbf{y}) \exp\left(\frac{1}{2} \ln\left(\frac{1 - \text{err}}{\text{err}} \right) (2 \times \mathbb{1}_{(y \neq f(x))} - 1) \right) \\ &= w(x, \mathbf{y}) \exp\left(\ln\left(\frac{1 - \text{err}}{\text{err}} \right) \mathbb{1}_{(y \neq f(x))} - \frac{1}{2} \ln\left(\frac{1 - \text{err}}{\text{err}} \right)) \right) \\ &= w(x, \mathbf{y}) \exp\left(\ln\left(\frac{1 - \text{err}}{\text{err}} \right) \mathbb{1}_{(y \neq f(x))} \right) \exp\left(-\frac{1}{2} \ln\left(\frac{1 - \text{err}}{\text{err}} \right) \right). \end{aligned}$$

Como $\exp\left(-\frac{1}{2} \ln\left(\frac{1-\text{err}}{\text{err}}\right)\right)$ no varía respecto a x ni y entonces actúa como una constante que se aplica de igual manera a cada peso. Por lo tanto, la actualización de los pesos es equivalente a

$$w(x, \mathbf{y}) \leftarrow w(x, \mathbf{y}) \exp\left(\ln\left(\frac{1 - \text{err}}{\text{err}} \right) \mathbb{1}_{(y \neq f(x))} \right).$$

De esta forma encontramos una explicación formal sobre cada una de las actualizaciones dentro del algoritmo de AdaBoost discreto. □

Corolario 1. *Al final de cada iteración del algoritmo, la actualización de los pesos hace que el error ponderado del últimо clasificador sea de* 50 %

Demostración. Se sigue directamente del hecho que c^* cumple que $E_w[e^{-c^* \mathbf{y} F(x)} \mathbf{y} F(x)] = 0$. Los términos positivos corresponden a los datos clasificados correctamente y los términos negativos a los datos clasificados de forma incorrecta, como la esperanza ponderada es igual a 0 entonces el peso de los datos clasificados correctamente es igual al de los datos clasificados erróneamente. □

Para continuar, la idea es expandir el espacio de funciones con el cual se está trabajando. Hemos trabajado con funciones que clasifican como -1 o 1. La idea ahora es usar funciones que tomen valores reales. Al igual que para AdaBoost Discreto, buscamos una forma de mejorar el estimador que teníamos previamente añadiéndole una nueva función.

En cada iteración del algoritmo de Adaboost Discreto tenemos que el tamaño de la actualización c^* es igual para todos los $x \in X$. A diferencia del anterior, el algoritmo de Adaboost Real puede tomar diferentes valores para cada $x \in X$, lo que se traduce en ciertos casos en una convergencia más rápida del algoritmo. A continuación vamos a ver el origen de las actualizaciones presentes en cada iteración de Adaboost Real.

Teorema 1.1.6. *El algoritmo de AdaBoost Real crea un modelo de regresión logística aditivo haciendo aproximaciones en cada iteración para minimizar* $J(F)$.

Supongamos que tenemos un modelo F y queremos encontrar una función $f : X \to \mathbb{R}$ tal que $J(F + f) < J(F)$. Para esto, vamos a minimizar $J(F(x) + f(x))$ en cada x. Es decir, para cada $x \in X$ buscamos $z \in \mathbb{R}$ que minimice $J(F(x) + z)$.

$$\begin{aligned} J(F(x) + z)) &= E[e^{-\mathbf{y}F(x)}e^{-yz}|x] \\ &= e^{-z}e^{-F(x)}P(\mathbf{y} = 1|x) + e^{z}e^{F(x)}P(\mathbf{y} = -1|x) \\ &= e^{-z}P_w(\mathbf{y} = 1|x) + e^{z}P_w(\mathbf{y} = -1|x). \end{aligned}$$

Al igual que para el algoritmo discreto, la función de pesos viene dada por $w(x, \mathbf{y}) = e^{-\mathbf{y}F(x)}$. Si derivamos $J(F(x) + z))$ con respecto a z e igualamos a 0 nos queda

$$\begin{aligned} \frac{\partial J(F(x) + z)}{\partial z} &= 0 \\ -e^{-z}P_w(\mathbf{y} = 1|x) + e^{z}P_w(\mathbf{y} = -1|x) &= 0 \\ e^{z}P_w(\mathbf{y} = -1|x) &= e^{-z}P_w(\mathbf{y} = 1|x) \\ e^{2z} &= \frac{P_w(\mathbf{y} = 1|x)}{P_w(\mathbf{y} = -1|x)} \end{aligned}$$

$$z = \frac{1}{2} \ln \left(\frac{P_w(\mathbf{y} = 1|x)}{P_w(\mathbf{y} = -1|x)} \right).$$

De esta forma podemos definir $f(x)$ en cada punto como:

$$f(x) = \frac{1}{2} \ln \left(\frac{P_w(\mathbf{y} = 1|x)}{P_w(\mathbf{y} = -1|x)} \right).$$

Para mantener la función de pesos como $w(x, \mathbf{y}) = e^{-\mathbf{y}F(x)}$ tenemos que actualizarla de la siguiente manera

$$w(x, \mathbf{y}) \leftarrow w(x, \mathbf{y}) e^{-\mathbf{y}f(x)}.$$

Corolario 2. *Si $F(x)$ es óptimo entonces tenemos que la esperanza condicional ponderada de y es nula.*

Dem: Si $F(x)$ es óptima entonces

$$\frac{\partial J(F(x))}{F(x)} = E[-e^{-\mathbf{y}F(x)} y] = E_w[-y] = 0.$$

Lo que nos dice este corolario es que cuando $F(x)$ es óptimo los pesos de los datos mal clasificados es igual para ambas clases y al ser $F(x)$ el óptimo también tenemos que debe ser pequeño. Si no fuera nula entonces todavía se podría mejorar la estimación dada por $F(x)$. Se sigue directamente de este corolario que si empezamos con $F(x) = 0$ y conocemos la función de probabilidad conjunta, entonces el algoritmo tendría una única iteración porque después de la primera iteración tendríamos directamente que $P_w(\mathbf{y} = 1|x) = P_w(\mathbf{y} = -1|x)$.

Para continuar, vamos a ver dos aproximaciones diferentes de Boosting en el mismo contexto. A diferencia de los algoritmos de Adaboost, estos utilizan el método Newton para minimizar la función objetivo.

1.1.2. LogitBoost y GentleBoost

Una pregunta interesante que surge al estudiar estos algoritmos es la escogencia de la función objetivo a minimizar y la forma de optimizarlas. El algoritmo de LogitBoost optimiza una función objetivo distinta a $E[e^{-\mathbf{y}F(x)}]$ y aproxima los valores óptimos usando el método de Newton en cada iteración. Para contrastar y tener un buen punto de comparación, el algoritmo de GentleBoost optimiza $E[e^{-\mathbf{y}F(x)}]$ usando el método de Newton.

Con el fin de entender el transfondo de la función objetivo de LogitBoost definimos $y^* = (y+1)/2$ que toma los valores 0, 1 y además tenemos que la probabilidad de que $x \in X$ pertenezca a la clase $\mathbf{y} = 1$ es

$$p(x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}} = \frac{e^{2F(x)}}{1 + e^{2F(x)}}.$$

Al igual que en (1.1) obtenemos el modelo logístico usual, la idea principal es ver el problema de encontrar la función F de forma local para cada x. De esta forma, buscamos el estimador de log-verosimilitud $L(y^*; p(x))$ para cada $x \in X$. En este caso, tenemos una única observación y^* para cada x y una distribución de Bernoulli con parámetro p(x).

El estimador de log-verosimilitud es

$$\begin{aligned} L(y^*; p(x)) &= \log(p(x)^{y^*}(1-p(x))^{1-y^*}), \\ L(y^*; p(x)) &= y^* \log(p(x)) + (1-y^*)\log((1-p(x))). \end{aligned}$$

Lema 1.1.7. *Tenemos que* $L(y^*; p(x)) = (2y^*F(x) - \log(1+e^{2F(x)}))$

Demostración. Para empezar veamos que se cumple cuando $y^* = 0$,

$$\begin{aligned} L(0; p(x)) &= \log(1-p(x)), \\ L(0; p(x)) &= \log\left(1 - \frac{e^{2F(x)}}{1+e^{2F(x)}}\right), \end{aligned}$$

$$L(0; p(x)) = \log\left(\frac{1}{1+e^{2F(x)}}\right),$$
$$L(0; p(x)) = -\log(1+e^{2F(x)}).$$

Solo nos falta verificar que se cumple para $y^* = 1$,

$$L(1; p(x)) = \log(p(x)),$$
$$L(1; p(x)) = \log\left(\frac{e^{2F(x)}}{1+e^{2F(x)}}\right),$$
$$L(1; p(x)) = \log(e^{2F(x)}) - \log(1+e^{2F(x)}),$$
$$L(1; p(x)) = 2F(x) - \log(1+e^{2F(x)}).$$

□

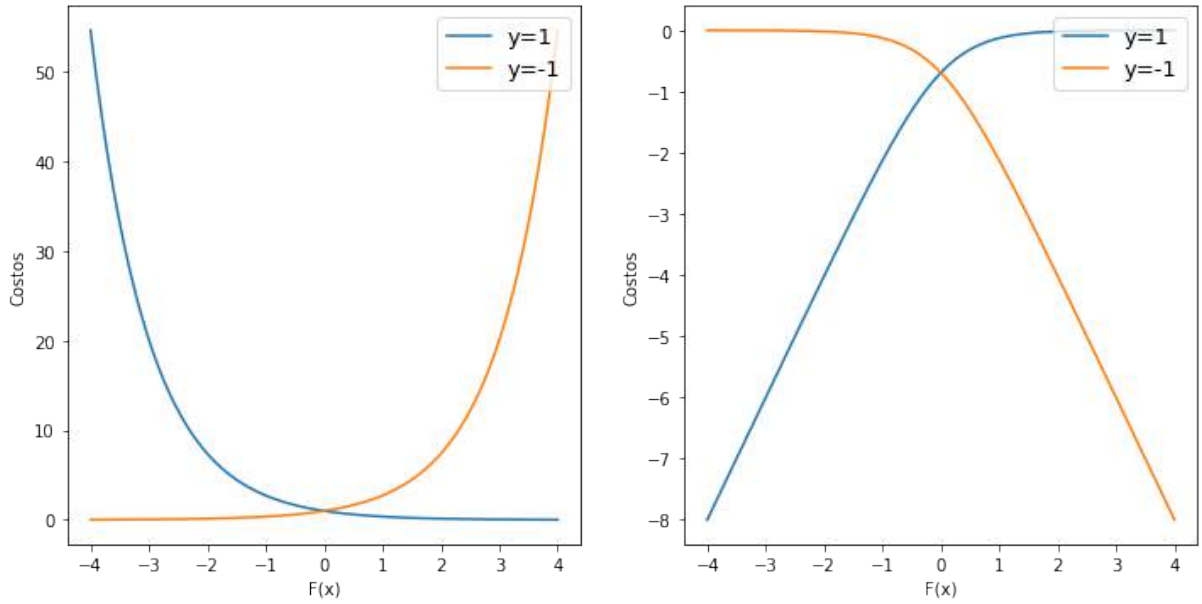

Figura 1.1: Valores de las funciones de costos en función de $F(x)$. A la izquierda se encuentran los valores de $e^{-yF(x)}$ y a la derecha los valores de $L(y^*; p(x))$

En la figura vemos claramente porque se intenta minimizar el criterio exponencial pero se busca maximizar la log-verosimilitud. Además, constatamos que el valor absoluto del costo crece de forma exponencial en el primero y de forma lineal en el segundo.

Teorema 1.1.8. *El algoritmo de LogitBoost utiliza actualizaciones mediante el método de Newton para ajustar un modelo logístico aditivo simétrico maximizando la log-verosimilitud.*

Demostración. Vamos a utilizar el lema 1.1.7 a la hora de calcular las derivadas de $L(y^*; p(x))$ para poder simplificar los cálculos. Para simplificar la notación vamos denotar la función de costo de un modelo F como

$$El(F) = E[2y^*F(x) - log(1 + e^{2F(x)})].$$

Supongamos que tenemos un modelo F y queremos encontrar una función $f : X \to \mathbb{R}$ de tal forma que $El(F + f) \geq El(F)$. Para esto vamos a condicionar con respecto a x y calcular las primeras dos derivadas de $El(F + f)$ en $f(x) = 0$,

$$\begin{aligned} s(x) &= \frac{\partial El(F(x) + f(x))}{\partial f(x)}\Big|_{f(x)=0} \\ &= \frac{\partial E[2y^*(F(x) + f(x)) - log(1 + e^{2(F(x)+f(x))}))|x]}{\partial f(x)}\Big|_{f(x)=0} \\ &= E\left[2y^* - \frac{2e^{2F(x)}}{1 + e^{2F(x)}}\Big|x\right] \\ &= 2E[y^* - p(x)|x], \end{aligned}$$

$$\begin{aligned} H(x) &= \frac{\partial^2 El(F(x) + f(x))}{\partial^2 f(x)}\Big|_{f(x)=0} \\ &= \frac{\partial s(x)}{\partial f(x)}\Big|_{f(x)=0} \\ &= 2E\left[-\frac{2e^{2F(x)}}{(1 + e^{2F(x)})^2}\Big|x\right] \end{aligned}$$

$$= -4E\left[\frac{e^{2F(x)}}{(1+e^{2F(x)})}\cdot\frac{1}{(1+e^{2F(x)})}\Big|x\right]$$
$$= -4E[p(x)(1-p(x))|x].$$

Se sigue que la actualización de Newton viene dada por

$$\begin{aligned} F(x) \leftarrow & F(x) - H(x)^{-1}s(x) \\ = & F(x) + \frac{E[y^* - p(x)|x]}{2E[p(x)(1-p(x))|x]} \\ = & F(x) + \frac{1}{2}E_w\left[\frac{y^* - p(x)}{p(x)(1-p(x))}\Big|x\right]. \end{aligned}$$

donde $w(x) = p(x)(1-p(x))$ □

El algoritmo resultante es el dado en 3.

De forma similar al Algoritmo de LogitBoost, queremos entender el comportamiento de Boosting si usamos el método de Newton para optimizar la función $E[e^{-\mathbf{y}F(x)}]$. Veremos que las actualizaciones resultantes son acotadas. Esto reduce la variación del modelo y de ahí mismo viene el nombre de GentleBoost.

Teorema 1.1.9. *El algoritmo de GentleBoost utiliza actualizaciones mediante el método de Newton para minimizar* $E[e^{-\mathbf{y}F(x)}]$.

Demostración. Al igual que para el algoritmo de LogitBoost, calculamos las dos primeras derivadas de $E[e^{-y(F(x)+f(x))}]$

$$s(x) = \frac{\partial E[e^{-y(F(x)+f(x))}|x]}{\partial f(x)}\Big|_{f(x)=0} = E[-e^{-yF(x)}y|x],$$
$$H(x) = \frac{\partial^2 E[e^{-y(F(x)+f(x))}|x]}{\partial^2 f(x)}\Big|_{f(x)=0} = E[e^{-yF(x)}|x].$$

Algoritmo 3 LogitBoost

Inicializar: Se define $F(x) = 0$

Inicializar: A cada uno de los datos se le asigna un peso $w_i = 1/N$ y una probilidad estimada $p(x_i) = 1/2$

Para m desde 1 hasta M **hacer**

Para cada x_i calcular el valor teórico correspondiente:

$$z_i = \frac{y^* - p(x_i)}{p(x_i)(1 - p(x_i))}.$$

Para cada x_i calcular su peso correspondiente $w_i = p(x_i)(1 - p(x_i))$.

Ajustar la función $f_m(x)$ por medio de una regresión por mínimos cuadrados ponderados de los z_i a los x_i usando los pesos w_i.

Actualizar el modelo $F(x) \leftarrow F(x) + \frac{1}{2} f_m(x)$.

Actualizar las probabilidades de cada x_i, $p(x) \leftarrow e^{F(x)}/(e^{F(x)} + e^{-F(x)})$.

fin Para

Retornar Retornar el clasificador dado por $\text{sgn}[F(x)] = \text{sgn}\left[\sum_{m=1}^{M} f_m(x)\right]$.

Por lo tanto la actualización en cada iteración es

$$\begin{aligned} F(x) \leftarrow & F(x) + \frac{E[-e^{-yF(x)}y|x]}{E[e^{-yF(x)}|x]} \\ = & F(x) + E_w[y|x]. \end{aligned}$$

donde $w(x, \mathbf{y}) = e^{-yF(x)}$. □

A diferencia de los demás algoritmos presentados previamente, GentleBoost es mucho más conservador que los otros. Entre cada iteración la diferencia de valores de $F(x)$ para un x fijo es menor o igual a 1. Esto permite que tenga un buen rendimiento y por lo general no tenga problemas de estabilidad.

Algoritmo 4 GentleBoost

Inicializar: Se define $F(x) = 0$.

Inicializar: A cada uno de los datos se le asigna un peso $w_i = 1/N$.

Para m desde 1 hasta M **hacer**

Ajustar la función $f_m(x)$ por medio de una regresión por mínimos cuadrados ponderados de los y_i a los x_i usando los pesos w_i.

Actualizar el modelo $F(x) \leftarrow F(x) + f_m(x)$.

Actualizar los pesos de cada x_i, $w_i \leftarrow w_i \exp(-\mathbf{y_i} f_m(x_i))$.

fin Para

Retornar Retornar el clasificador dado por $\text{sgn}[F(x)] = \text{sgn}\left[\sum_{m=1}^{M} f_m(x)\right]$.

1.2. Aproximación Abstracta del Problema

A continuación vamos a presentar algunos resultados presentados en [MBBF00] con el fin de tener una forma general de crear algoritmos de Boosting. Consideramos que tenemos datos de la forma (x, y) generados de forma aleatoria con respecto a una distribución de probabilidad desconocida D en $X \times Y$ donde por lo general $X \subset \mathbb{R}^N$ y Y es un conjunto discreto o un subconjunto de $\mathbb{R}$.
La idea, al igual que en la sección anterior, es obtener algoritmos que sean combinaciones lineales de clasificadores y tengan la forma $\text{sgn}(F(x))$, donde

$$F(x) = \sum_{m=1}^{M} w_m f_m(x),$$

los $f_m : X \mapsto \{\pm 1\}$ son clasificadores base de la clase $\mathcal{F}$ y $w_m \in \mathbb{R}$ representan los pesos asignados a cada uno de estos clasificadores.
Definimos el margen de un dato $(x, \mathbf{y})$ con respecto al clasificador $\text{sgn}(F(x))$ como el producto $yF(x)$. Dado un conjunto de datos $S = \{(x_1, \mathbf{y}_1), (x_2, \mathbf{y}_2), \ldots, (x_k, \mathbf{y}_k)\}$ con k parejas generadas de forma aleatoria siguiendo la distribución D. La idea principal de los algoritmos es construir una combinación de clasificadores con la forma de $F(x)$ tal que $P_D(\text{sgn}(F(x) \neq y)$ sea pequeña. Teniendo en cuenta que D es desconocida, se usan los datos en S para encontrar una función que minimice el costo medio de una función de costos C del margen. Esto es que, para un conjunto de datos S y una función de costos C, queremos encontrar F^* tal que:

$$H(G) = \frac{1}{k} \sum_{i=1}^{k} C(G(x_i)\mathbf{y}_i),$$

$$F^* = \underset{G \in \mathcal{F}}{\arg\min} \quad H(G).$$

En este caso, $H : \mathcal{F} \mapsto \mathbb{R}$ representa el funcional de costos.

Una de las formas de producir combinaciones de clasificadores con pesos que optimiza F es por medio del descenso de gradiente en un espacio de funciones. Podemos

escribir una manera abstracta de tratar este descenso de gradiente con un producto interno que se acomode al espacio que usamos.

Para lo anterior, vamos a ver a todas las funciones $f \in \mathcal{F}$ y sus combinaciones F como elementos de un espacio vectorial provisto con un producto interno $(\mathcal{S}, \langle \cdot, \cdot \rangle)$. Para este problema, tenemos que $\mathcal{S}$ es un espacio vectorial de funciones que contiene a todas las combinaciones lineales de funciones en $\mathcal{F}$, el cual notamos como $\text{lin}(\mathcal{F})$ y su producto interno lo definimos como

$$\langle F, G \rangle := \sum_{i=1}^{k} F(x_i)G(x_i), \tag{1.2}$$

para todos $F, G \in \text{lin}(\mathcal{F})$.

1.2.1. Definición del gradiente

Ahora supongamos que tenemos una función $F \in \text{lin}(\mathcal{F})$ y queremos encontrar una función $f \in \mathcal{F}$ tal que el costo $H(F + \epsilon f)$ sea menor que $H(F)$ para un valor pequeño de ϵ. Teniendo en cuenta que estamos considerando funciones del tipo $F : X \subset \mathbb{R}^N \mapsto \mathbb{R}$, estamos buscando la dirección f tal que $H(F + \epsilon f)$ decrezca lo más rápido posible. Por lo tanto, la dirección que estamos buscando corresponde a menos la derivada del funcional H en F, $-\nabla H(F)$, con

$$\nabla H(F)(x) := \left. \frac{\partial H(F + t\mathbb{1}_x)}{\partial t} \right|_{t=0}. \tag{1.3}$$

Tenemos que considerar que la anterior derivada se define de forma discreta y para cada x que no pertenezca a mis datos esta va a ser 0.
Veamos cuanto vale $\nabla H(F)(x)$ para un x fijo.

$$\nabla H(F)(x) = \lim_{t \to 0} \frac{H(F + t\mathbb{1}_x)(x) - H(F)(x)}{t}$$

$$= \lim_{t\to 0} \frac{\frac{1}{k}\sum_{i=1}^{k} C((F+t\mathbb{1}_x)(x_i)\mathbf{y}_i) - \frac{1}{k}\sum_{i=1}^{k} C(F(x_i)\mathbf{y}_i)}{t}$$
$$= \lim_{t\to 0} \frac{\sum_{i=1}^{k} C((F+t\mathbb{1}_x)(x_i)\mathbf{y}_i) - C(F(x_i)\mathbf{y}_i)}{kt}.$$

Si $x \neq x_j$ para todo $j = 1, ..., k$

$$\begin{aligned}\nabla H(F)(x) &= \lim_{t\to 0} \frac{\sum_{i=1}^{k} C(F(x_i)\mathbf{y}_i + t\mathbb{1}_x(x_i)\mathbf{y}_i) - C(F(x_i)\mathbf{y}_i)}{kt} \\ &= \lim_{t\to 0} \frac{\sum_{i=1}^{k} C(F(x_i)\mathbf{y}_i) - C(F(x_i)\mathbf{y}_i)}{kt} \\ &= 0.\end{aligned}$$

Si $x = x_j$ para algún $j = 1, ..., k$

$$\begin{aligned}\nabla H(F)(x) &= \lim_{t\to 0} \frac{C((F+t\mathbb{1}_x)(x_j)\mathbf{y}_j) - C(F(x_j)\mathbf{y}_j)}{kt} \\ &= \frac{1}{k}\lim_{t\to 0} \frac{C((F+t\mathbb{1}_x)(x_j)\mathbf{y}_j) - C(F(x_j)\mathbf{y}_j)}{t} \\ &= \frac{1}{k}\lim_{t\to 0} \frac{C(F(x_j)\mathbf{y}_j + t\mathbf{y}_j) - C(F(x_j)\mathbf{y}_j)}{t} \\ &= \frac{1}{k}\lim_{t\to 0} \frac{C((F(x_j)+t)\mathbf{y}_j) - C(F(x_j)\mathbf{y}_j)}{t} \\ &= \frac{1}{k}\mathbf{y}_j C'(F(x_j)\mathbf{y}_j).\end{aligned}$$

Cabe resaltar que se cumple la expansión en primer orden con respecto a ϵ de la función H.

Lema 1.2.1. *Se cumple que* $H(F+\epsilon f) = H(F) + \epsilon\langle \nabla H(F), f\rangle + o(\epsilon^2)$.

Demostración. Primero reescribimos la ecuación como

$$\frac{H(F+\epsilon f) - H(F)}{\epsilon} = \langle \nabla H(F), f\rangle + o(\epsilon)$$

Ahora podemos probar el lema,

$$\begin{aligned} H(F+\epsilon f) - H(F) &= \frac{1}{k}\sum_{i=1}^{k} C((F(x_i)+\epsilon f(x_i))(\mathbf{y}_i)) - \frac{1}{k}\sum_{i=1}^{k} C(F(x_i)\mathbf{y}_i) \\ &= \frac{1}{k}\sum_{i=1}^{k} C((F(x_i)+\epsilon f(x_i))(\mathbf{y}_i)) - C(F(x_i)\mathbf{y}_i). \end{aligned}$$

Dividimos ambos lados por ϵ

$$\begin{aligned} \frac{H(F+\epsilon f) - H(F)}{\epsilon} &= \frac{1}{k}\sum_{i=1}^{k} \frac{C((F(x_i)+\epsilon f(x_i))(\mathbf{y}_i)) - C(F(x_i)\mathbf{y}_i)}{\epsilon}, \\ &= \frac{1}{k}\sum_{i=1}^{k} \frac{C((F(x_i)+\epsilon f(x_i))(\mathbf{y}_i)) - C(F(x_i)\mathbf{y}_i)}{\epsilon} \cdot \frac{f(x_i)}{f(x_i)} \end{aligned}$$

Tomamos límite cuando ϵ tiende a 0 y usamos el cambio de variable $h_i = \epsilon f(x_i)$ para cada término de la suma obtenemos,

$$\begin{aligned} \lim_{\epsilon\to 0} \frac{H(F+\epsilon f) - H(F)}{\epsilon} &= \lim_{\epsilon\to 0} \frac{1}{k}\sum_{i=1}^{k} \frac{C((F(x_i)+\epsilon f(x_i))(\mathbf{y}_i)) - C(F(x_i)\mathbf{y}_i)}{\epsilon f(x_i)} f(x_i) \\ &= \frac{1}{k}\sum_{i=1}^{k} \lim_{h_i\to 0} \frac{C((F(x_i)+h_i)(\mathbf{y}_i)) - C(F(x_i)\mathbf{y}_i)}{h_i} f(x_i) \\ &= \frac{1}{k}\sum_{i=1}^{k} \mathbf{y}_i C'(F(x_i)\mathbf{y}_i) f(x_i). \end{aligned}$$

Por otro lado, tenemos que,

$$\begin{aligned} \langle \nabla H(F), f\rangle &= \sum_{i=1}^{k} \nabla H(F)(x_i) f(x_i) \\ &= \sum_{i=1}^{k} \frac{1}{k}\mathbf{y}_j C'(F(x_j)\mathbf{y}_j) f(x_i) \end{aligned}$$

$$= \frac{1}{k} \sum_{i=1}^{k} \mathbf{y}_j C'(F(x_j)\mathbf{y}_j) f(x_i).$$

Finalmente obtenemos el resultado deseado:

$$\begin{aligned}
&\lim_{\epsilon \to 0} \frac{H(F + \epsilon f) - H(F)}{\epsilon} = \langle \nabla H(F), f \rangle \\
&\Rightarrow \frac{H(F + \epsilon f) - H(F)}{\epsilon} = \langle \nabla H(F), f \rangle + o(\epsilon) \\
&\Rightarrow H(F + \epsilon f) - H(F) = \epsilon \langle \nabla H(F), f \rangle + o(\epsilon^2) \\
&\Rightarrow H(F + \epsilon f) = H(F) + \epsilon \langle \nabla H(F), f \rangle + o(\epsilon^2).
\end{aligned}$$

□

Esto nos permite ver que $\nabla H(F)$ está bien definido en (1.3) y podemos usar el método de descenso de gradiente en este contexto.

1.2.2. Algoritmos de AnyBoost

Para empezar, vamos a utilizar lo anterior para crear el algoritmo de AnyBoost. Este toma un conjunto de funciones $\mathcal{F}$ y busca un combinación lineal F que minimice la función de costos $H(F)$. Para esto, buscan aproximar el valor negativo de la función dada por el gradiente $\nabla C(F)$. Lo único necesario es que la función $H : \text{lin}(\mathcal{F}) \to \mathbb{R}$ sea diferenciable.

Podemos resaltar que el algoritmo 5 no tiene ninguna restricción en particular. Para cada problema se pueden ajustar la función de costos, los posibles valores de Y y el tamaño del paso w_m. Además, notemos que el algoritmo para cuando $-\langle \nabla H(F_m), f_{m+1} \rangle \leq 0$. Por lo tanto, el algoritmo termina cuando la función f_{m+1} aumentaría el costo.

Para evitar el overfitting es posible reducir el espacio de funciones. En vez de traba-

Algoritmo 5 Algoritmo de AnyBoost

Entrada: Sea $\mathcal{L}(F)$ un clasificador débil que recibe una función $F \in \text{lin}(\mathcal{F})$ y retorna $f \in \mathcal{F}$ que intenta maximizar $-\langle \nabla H(F), f \rangle$.

Inicializar: Sea $F_0(x) = 0$,

Para m desde 1 hasta M **hacer**

$f_{m+1} \leftarrow \mathcal{L}(F_m)$.

Si $-\langle \nabla H(F_m), f_{m+1} \rangle \leq 0$ **entonces**

Retornar F_m.

fin Si

Escoger w_{m+1}.

Definir $F_{m+1} = F_m + w_{m+1} f_{m+1}$.

fin Para

Retornar F_{M+1}.

jar sobre $\text{lin}(\mathcal{F})$ podemos restringirnos a las combinaciones convexas de funciones en $\mathcal{F}$. Lo anterior lo logramos haciendo un pequeño cambio sobre la actualización en el algoritmo de AnyBoost. Vamos a redefinir la actualización en cada iteración como $F_{m+1} = \alpha F_m + (1-\alpha) f_{m+1}$ para $\alpha \in [0,1]$. Veamos exactamente cuanto está variando la función en cada iteración:

$$\begin{aligned} F_{m+1} - F_m &= \alpha F_m + (1-\alpha) f_{m+1} - F_m, \\ &= (1-\alpha)(f_{m+1} - F_m). \end{aligned}$$

Al igual que en el caso previo, se busca que $-\langle \nabla H(F_m), f_{m+1} - F_m \rangle$ sea estrictamente positivo para reducir la aproximación lineal del costo. Nos queda el siguiente algoritmo:

Algoritmo 6 Algoritmo de AnyBoost Convexo

Entrada: Sea $\mathcal{L}(F)$ un clasificador débil que recibe una función $F \in \text{lin}(\mathcal{F})$ y retorna $f \in \mathcal{F}$ que intenta maximizar $-\langle \nabla H(F), f - F \rangle$.

Inicializar: Sea $F_0(x) = 0$,

Para m desde 1 hasta M **hacer**

$f_{m+1} \leftarrow \mathcal{L}(F_m)$.

Si $-\langle \nabla H(F_m), f_{m+1} - F_m \rangle \leq 0$ **entonces**

Retornar F_m.

fin Si

Escoger w_{m+1}.

Definir $F_{m+1} = F_m + \frac{w_{m+1} f_{m+1}}{\sum_{i=0}^{m+1} |w_i|}$.

fin Para

Retornar F_{M+1}.

Ejemplo de Anyboost

Para aterrizar un poco los algoritmos presentados previamente, los vamos a poner en el contexto de AdaBoost discreto. Es decir, tomemos el conjunto $\mathcal{F} := \{f : \mathbf{X} \to \{\pm 1\}\}$ y tomemos una función de costos

$$H(F) := \frac{1}{k} \sum_{i=1}^{k} C\left(-\mathbf{y}_i F(x_i)\right) = \frac{1}{k} \sum_{i=1}^{k} e^{-\mathbf{y}_i F(x_i)}.$$

Si calculamos el gradiente, nos queda que

$$\nabla H(F)(x) = \begin{cases} 0, & \text{si } x \neq x_i, i = 1 \dots m, \\ -\frac{1}{k} \mathbf{y}_i e^{-\mathbf{y}_i F(x_i)}, & \text{si } x = x_i. \end{cases}$$

Con este podemos calcular el producto interno definido en (1.2):

$$-\langle \nabla H(F), f \rangle = \frac{1}{k} \sum_{i=1}^{k} \mathbf{y}_i f(x_i) e^{-\mathbf{y}_i F(x_i)}. \tag{1.4}$$

La idea es encontrar un f que maximice (1.4). Para esto, $f(x_i)$ siempre va a intentar tener el mismo signo de $\mathbf{y}_i$ y busca tener valores mucho mayores cuando $F(x_i)\mathbf{y}_i < 0$. Si tomamos el peso asignado a cada clasificador como

$$w_{m+1} = \frac{1}{2} ln \left(\frac{1 - \text{err}}{\text{err}} \right),$$

donde

$$\text{err} := \frac{\sum_{i=1}^{k} \mathbb{1}_{(\mathbf{y}_i \neq F_m(x_i))} e^{-(\mathbf{y}_i F_m(x_i))}}{\sum_{i=1}^{k} e^{-(\mathbf{y}_i F_m(x_i))}}. \tag{1.5}$$

Entonces nos queda el siguiente algoritmo:

Algoritmo 7 Algoritmo de Ejemplo

Entrada: Sea $\mathcal{L}(F)$ un clasificador débil que recibe una función $F \in \text{lin}(\mathcal{F})$ y retorna $f \in \mathcal{F}$ que intenta maximizar $-\langle \nabla H(F), f - F \rangle$.

Inicializar: Sea $F_0(x) = 0$,

Para m desde 1 hasta M **hacer**

$f_{m+1} \leftarrow \mathcal{L}(F_m)$,

Si $\sum_{i=1}^{k} \mathbf{y}_i f_{m+1}(x_i) e^{-\mathbf{y}_i F_m(x_i)} \leq 0$ **entonces**

Retornar F_m.

fin Si

Calcular w_{m+1}.

Definir $F_{m+1} = F_m + \frac{w_{m+1} f_{m+1}}{\sum_{i=0}^{m+1} |w_i|}$.

fin Para

Retornar F_{M+1}.

Es interesante ver la similitud entre este algoritmo y el de AdaBoost Discreto. Teniendo en cuenta que los pesos en el algoritmo de Adaboost Discreto vienen dados por $w(x, y) = e^{-yF(x)}$, es fácil ver que el error en (1.5) es equivalente a $E_w[\mathbb{1}_{y \neq F(x)}]$. Una de las diferencia radica en el modelo que entrenan en la iteración $m + 1$: AdaBoost Discreto busca ajustar los $\mathbf{y}_i$ a los x_i usando los pesos w_i, en cambio, este algoritmo busca ajustar los $\mathbf{y}_i e^{-\mathbf{y}_i F_m(x_i)}$ a los x_i. El primero entrena un algoritmo de clasificación y para el segundo se intenta predecir los valores $\mathbf{y}_i e^{-\mathbf{y}_i F_m(x_i)}$ por lo que se necesita un algoritmo de regresión. Para encontar una función en el espacio definido previamente, se aplica la función de signo a la predicción. Por consiguiente, vamos a tener que los pesos estarán implícitos dentro del entrenamiento de la función.
La mayor diferencia está en el criterio de parada del algoritmo. El algoritmo de AdaBoost Discreto siempre hace todas sus iteraciones, en cambio, el algoritmo presentado se detiene si el error ponderado es mayor o igual al 50 %.

Por otro lado, si tomamos los $w_m = 1$ para todo m y permitimos funciones que tomen valores en todos los reales, nos queda un algoritmo prácticamente igual a LogitBoost. Lo importante es darse cuenta que todas las actualizaciones de LogitBoost vienen acompañadas del factor 1/2 entonces podríamos multiplicar el modelo resultante por 2 y definir

$$p(x) = \frac{1}{1 + e^{F(x)}}.$$

Con esto, recordemos que los z_i en LogitBoost vienen dados por

$$z_i = \frac{y^* - p(x_i)}{p(x_i)(1 - p(x_i))}.$$

Si $y = 1$ entonces $y^* = 1$ y nos queda:

$$\begin{aligned} z_i &= \frac{1 - p(x_i)}{p(x_i)(1 - p(x_i))} \\ z_i &= \frac{1}{p(x_i)} \\ z_i &= 1 + e^{-F(x)}. \end{aligned}$$

Si $y = -1$ entonces $y^* = 0$ y nos queda:

$$\begin{aligned} z_i &= \frac{-p(x_i)}{p(x_i)(1-p(x_i))} \\ z_i &= \frac{-1}{1-p(x_i)} \\ z_i &= -(1+e^{F(x)}). \end{aligned}$$

El algoritmo generado busca predecir los valores $\mathbf{y}_i e^{-\mathbf{y}_i F_m(x_i)}$. La diferencia está en que en LogitBoost los z_i no toman valores en $[-1, 1]$.

Con esto terminamos nuestro análisis de los algoritmos de Boosting para clasificación de dos categorías.

1.3. Resultados

En esta sección se mostrarán los resultados obtenidos al probar los algoritmos descritos en la primera sección. Se realizaron varios experimentos con diferentes tipos de datos y clasificadores débiles. Es importante tener en cuenta que en la implementación de los algoritmos se hicieron pequeñas modificaciones para evitar problemas numéricos y mejorar el rendimiento de algunos algoritmos. Estos cambios se concentraban en acotar los valores que pueden generar los algoritmos para las siguientes iteraciones.

Para empezar, veremos el desempeño de los algoritmos para los siguientes datos en dimensión 2:

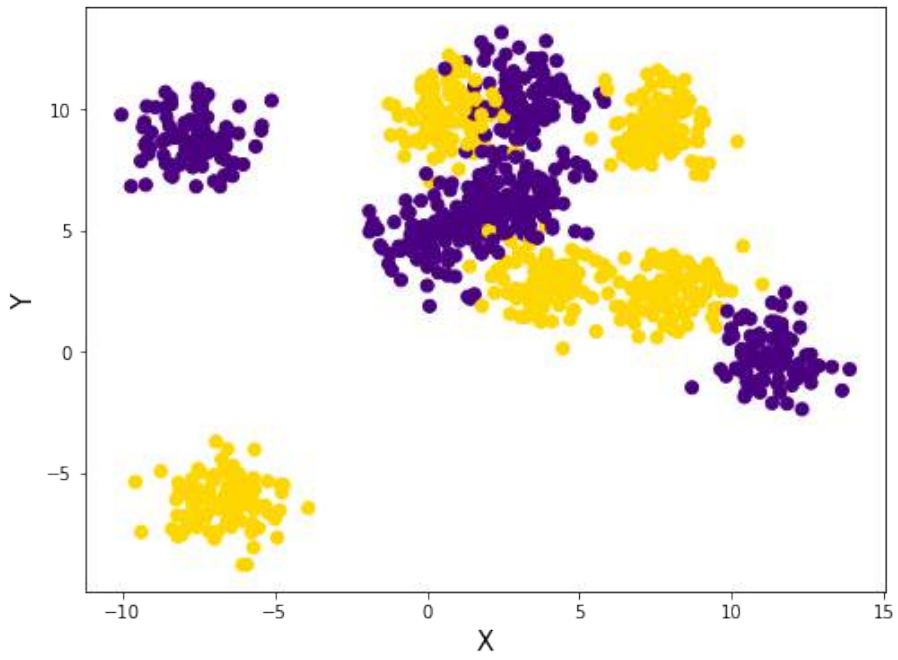

Figura 1.2: Se tomaron 10 nubes de datos y aleatoriamente se asignaron 5 nubes a cada clase. En total se tomaron 1000 datos

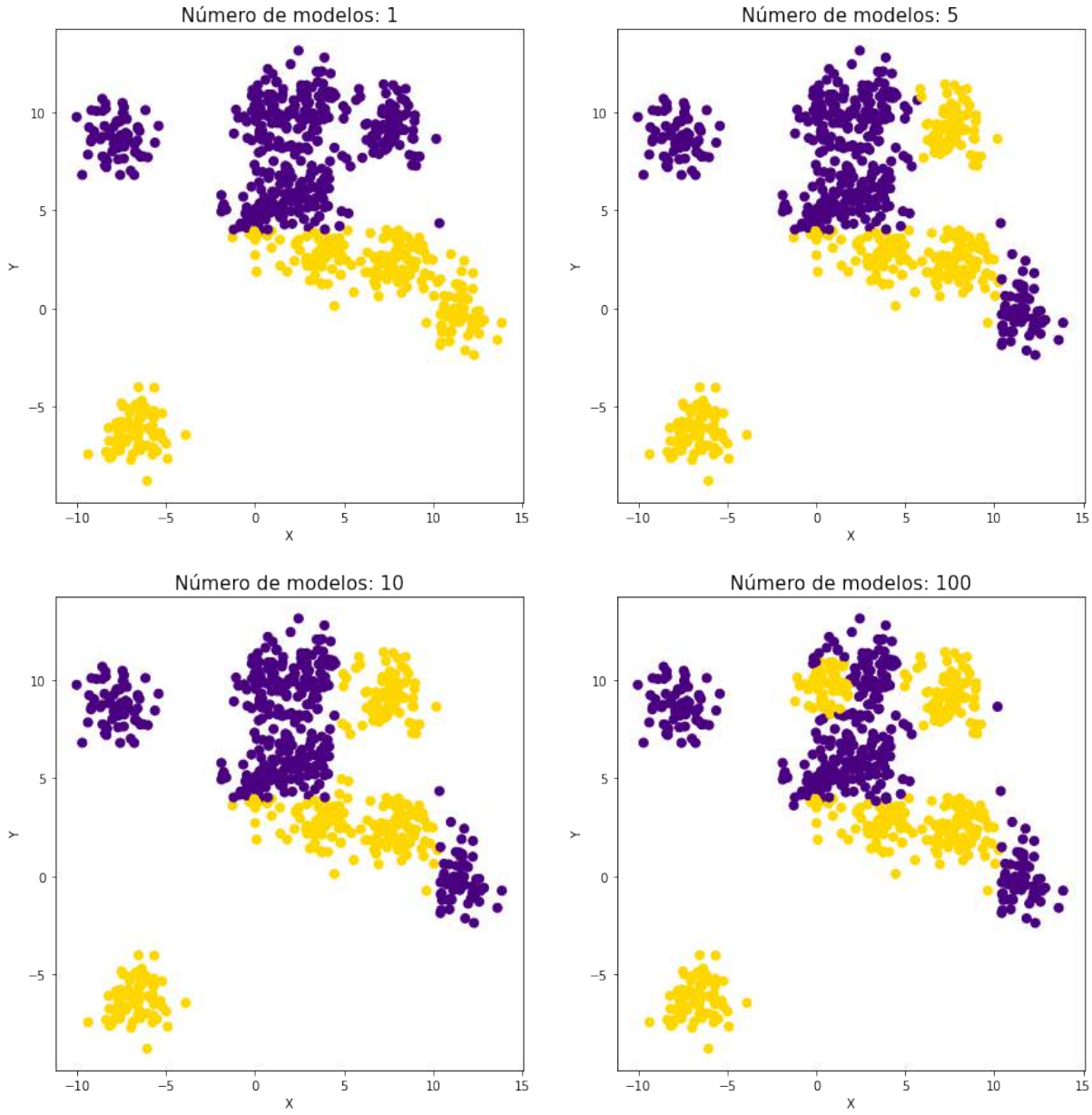

Figura 1.3: Predicciones del algoritmo de AdaBoost Discreto teniendo en cuenta únicamente los primeros 1,5,10 y 100 modelos. Usando como modelos de predicción árboles de decisión con profundidad máxima igual a 1.

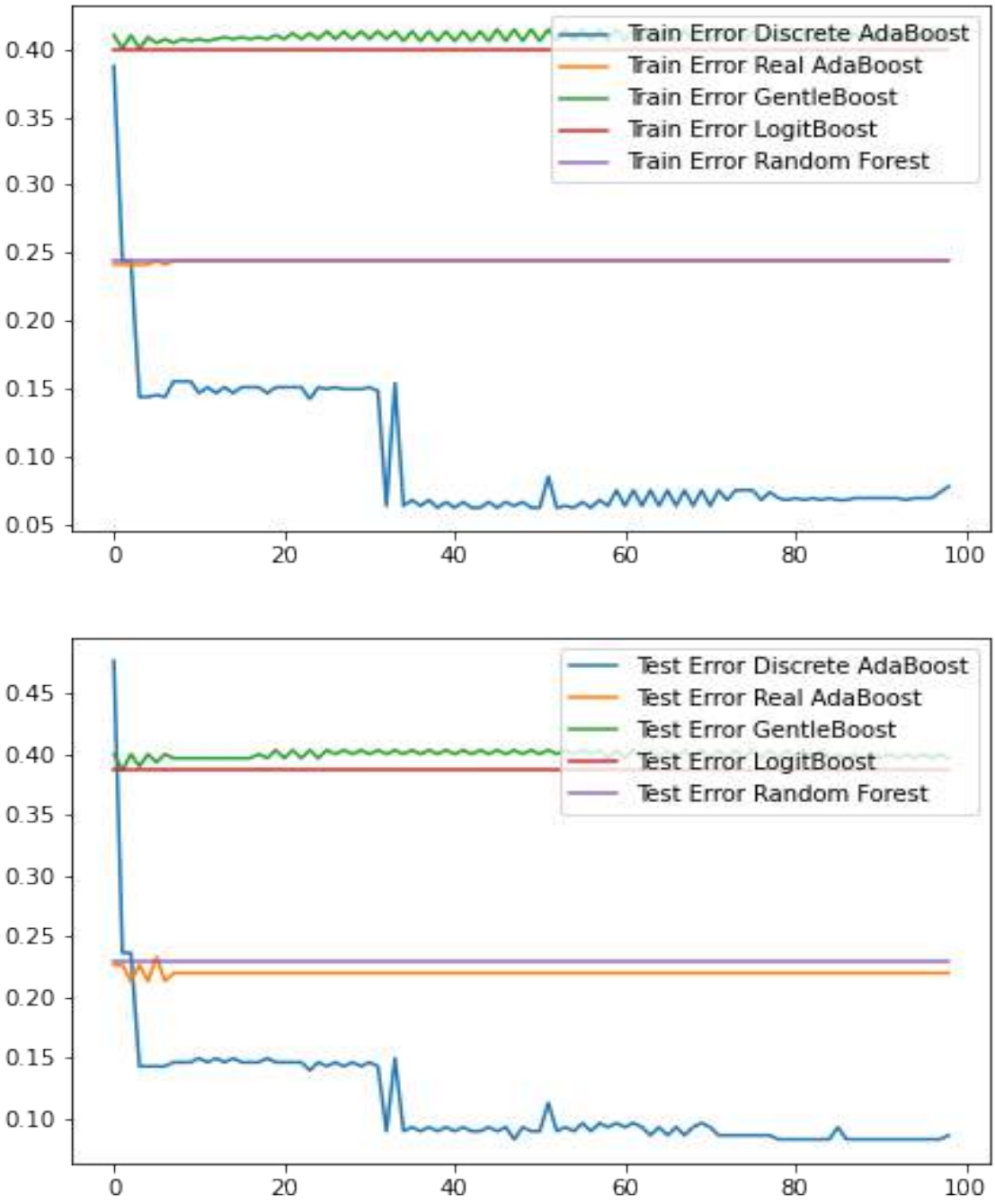

Figura 1.4: Resultados obtenidos al usar regresiones lineales para LogitBoost y GentleBoost y árboles de decisión 1 en los demás algoritmos.

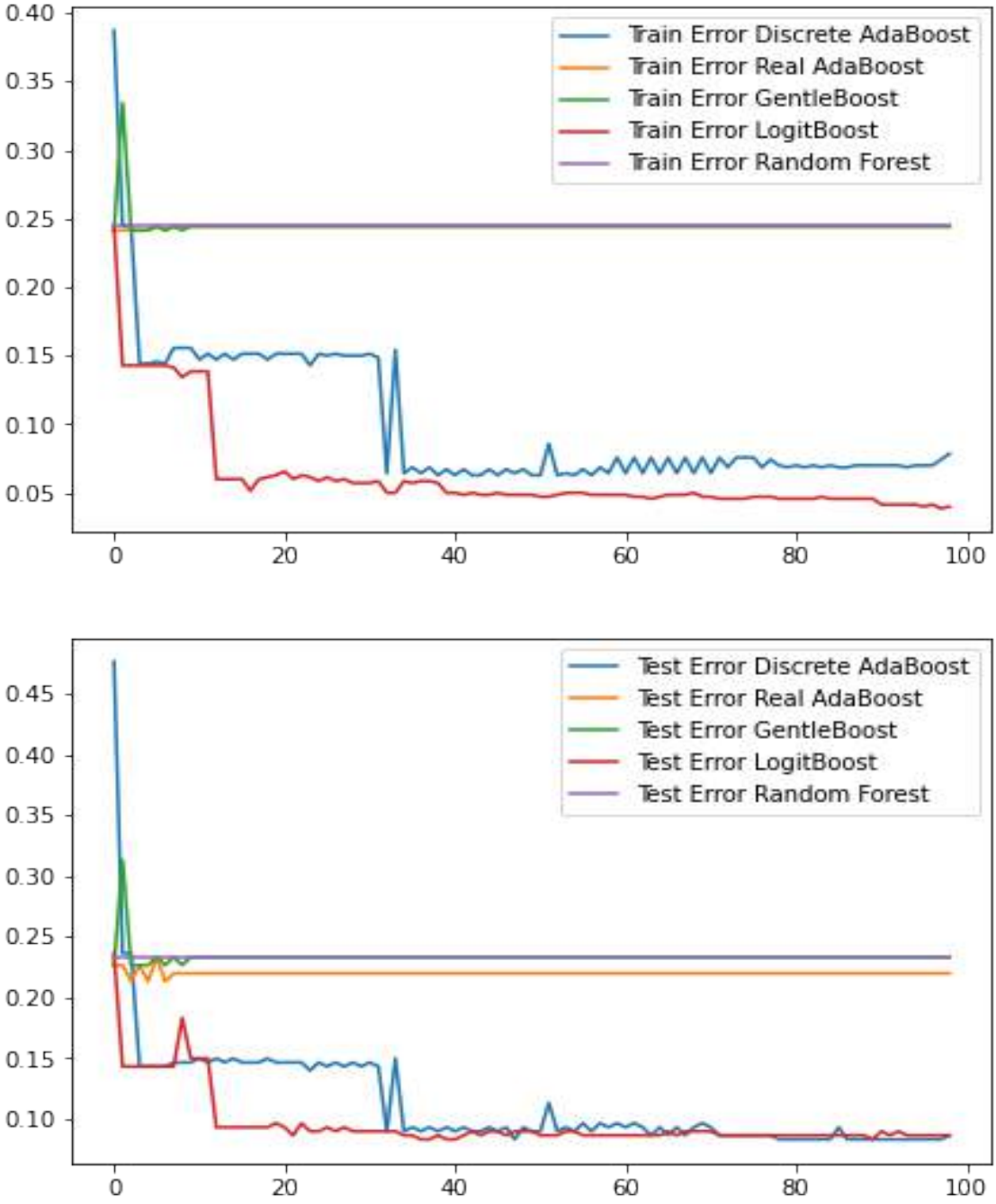

Figura 1.5: Resultados obtenidos al usar árboles de decisión con profundidad máxima igual a 1 en todos los algoritmos.

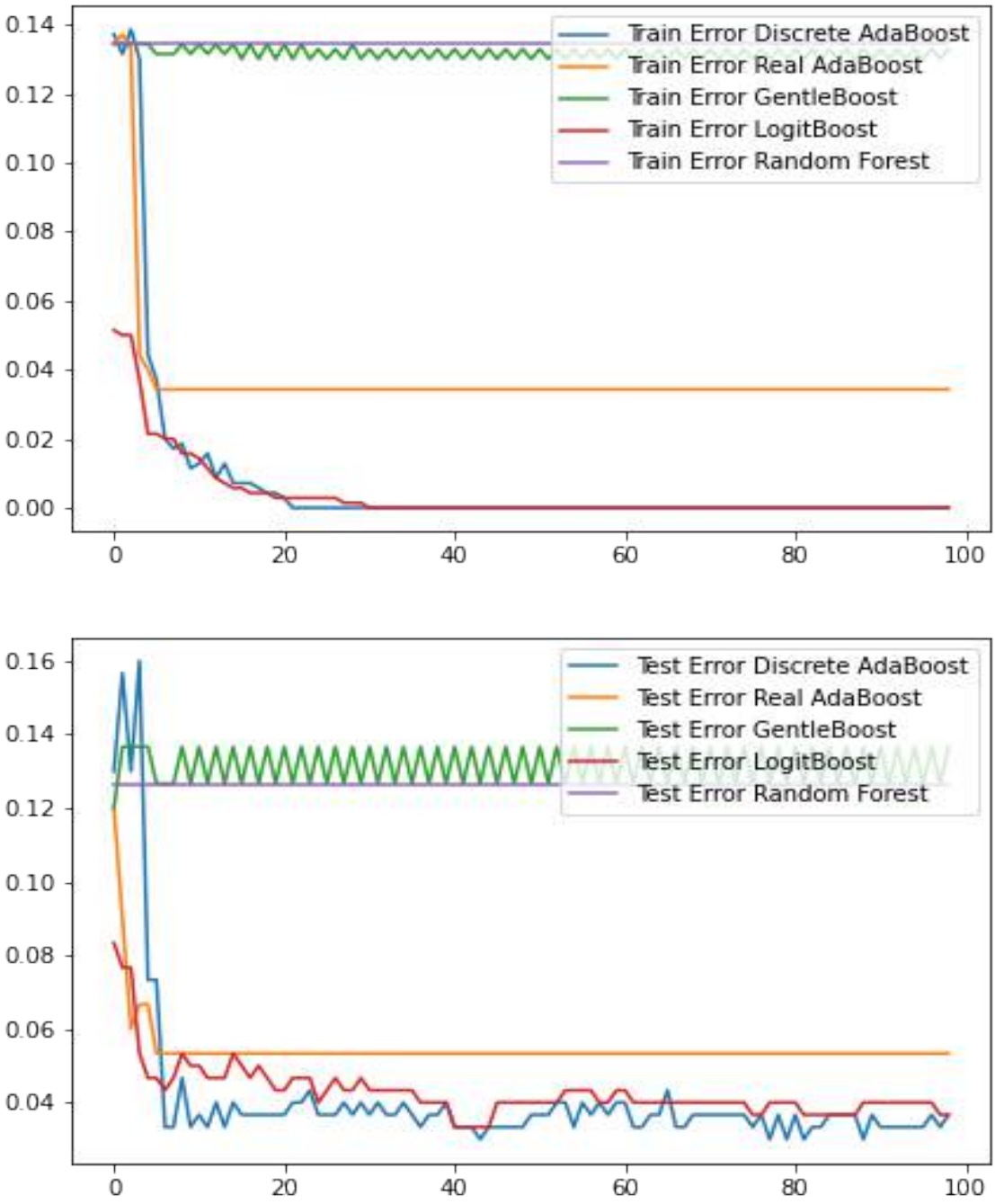

Figura 1.6: Resultados obtenidos al usar árboles de decisión con profundidad máxima igual a 3 en todos los algoritmos.

Los anteriores resultados muestran la importancia que tiene la familia de clasificadores que escogemos para construir nuestros modelos. Para los ejemplos estudiados los algoritmos de LogitBoost y Adaboost Discreto fueron los que obtuvieron mejores resultados. Además, comparando los algoritmos obtenidos con un Random Forest con 100 árboles vemos que en su mayoría, los resultados obtenidos son iguales o mejores.

Ahora generamos los datos de cada clase con una distribución normal bivariada. La primera está centrada en $(1, 1)$ y su varianza es la matriz identidad $\boldsymbol{I}$. La otra está centrada en $(3, 3)$ y su varianza es $2\boldsymbol{I}$. En este ejemplo no usamos el algoritmo de Random Forest como punto de comparación, en cambio,calculamos la función teórica presentada en 1.1.3.

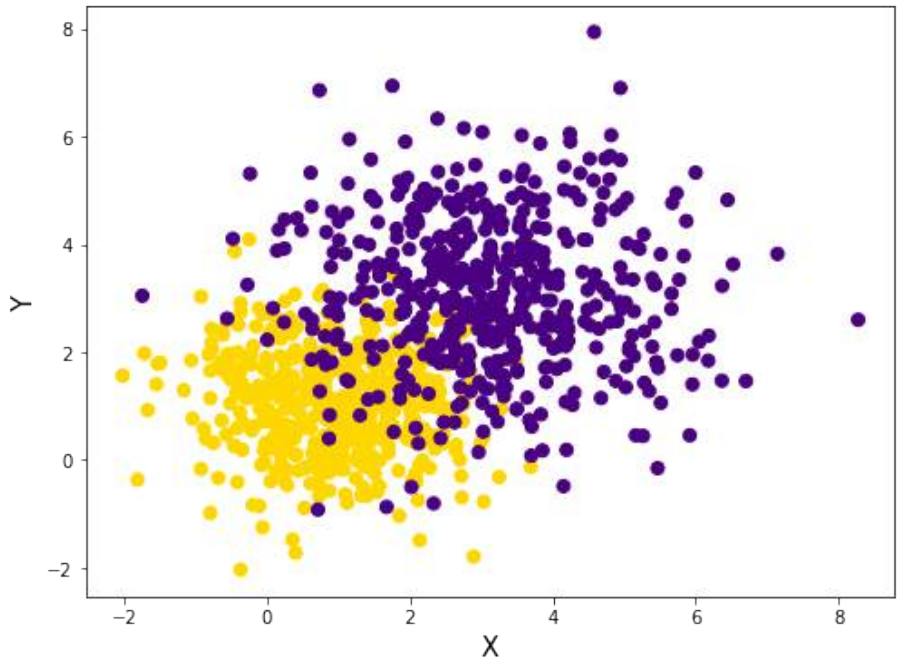

Figura 1.7: Datos generados con las distribuciones normales descritas previamente

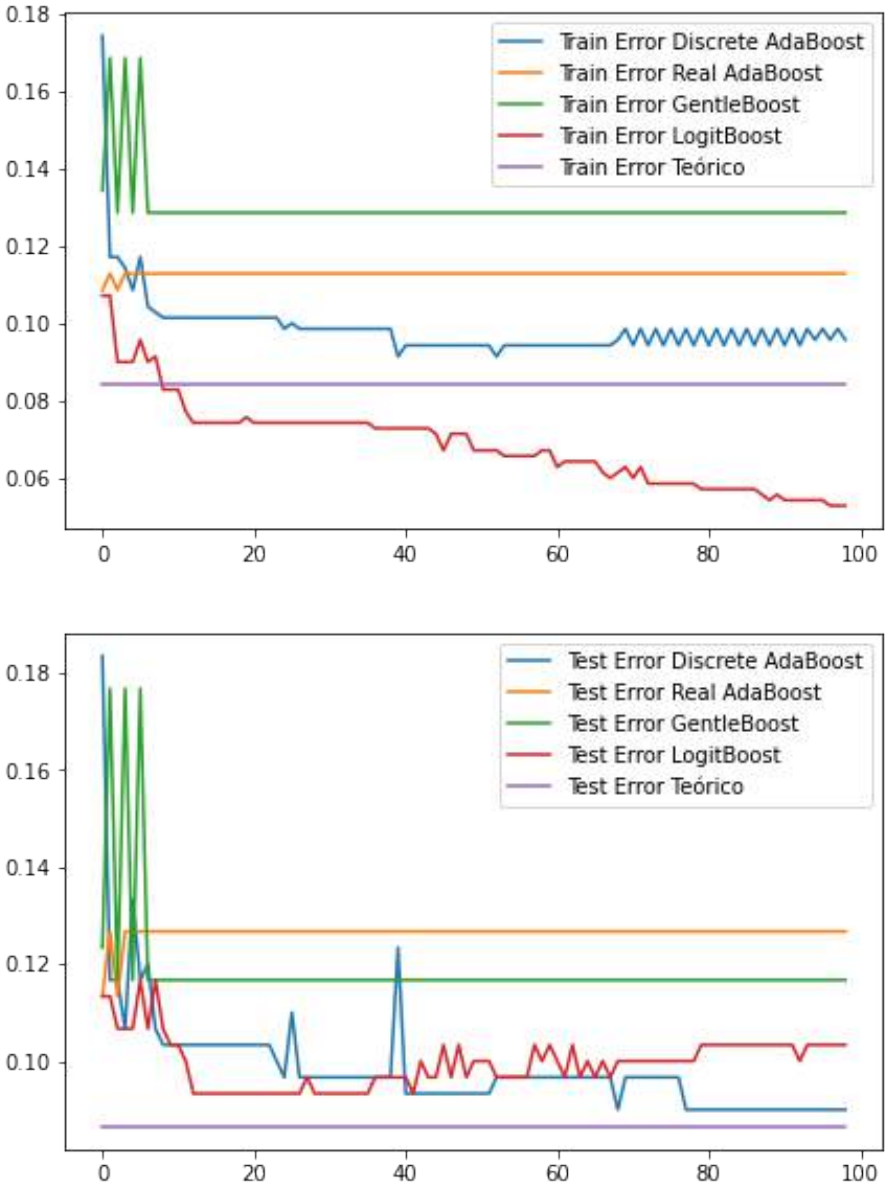

Figura 1.8: Resultados obtenidos al usar árboles de decisión con profundidad máxima igual a 1.

Vemos que Algunos algoritmos son capaces de obtener mejores resultados de entrenamiento que el clasificador teórico, pero, como era de esperarse el clasificador

teórico es mejor para predecir un conjunto de datos nuevo.
Por otro lado, si aumentamos la dimensión de los datos, el tiempo de ejecución de los algoritmos aumenta de forma lineal y se obtienen resultados similares.

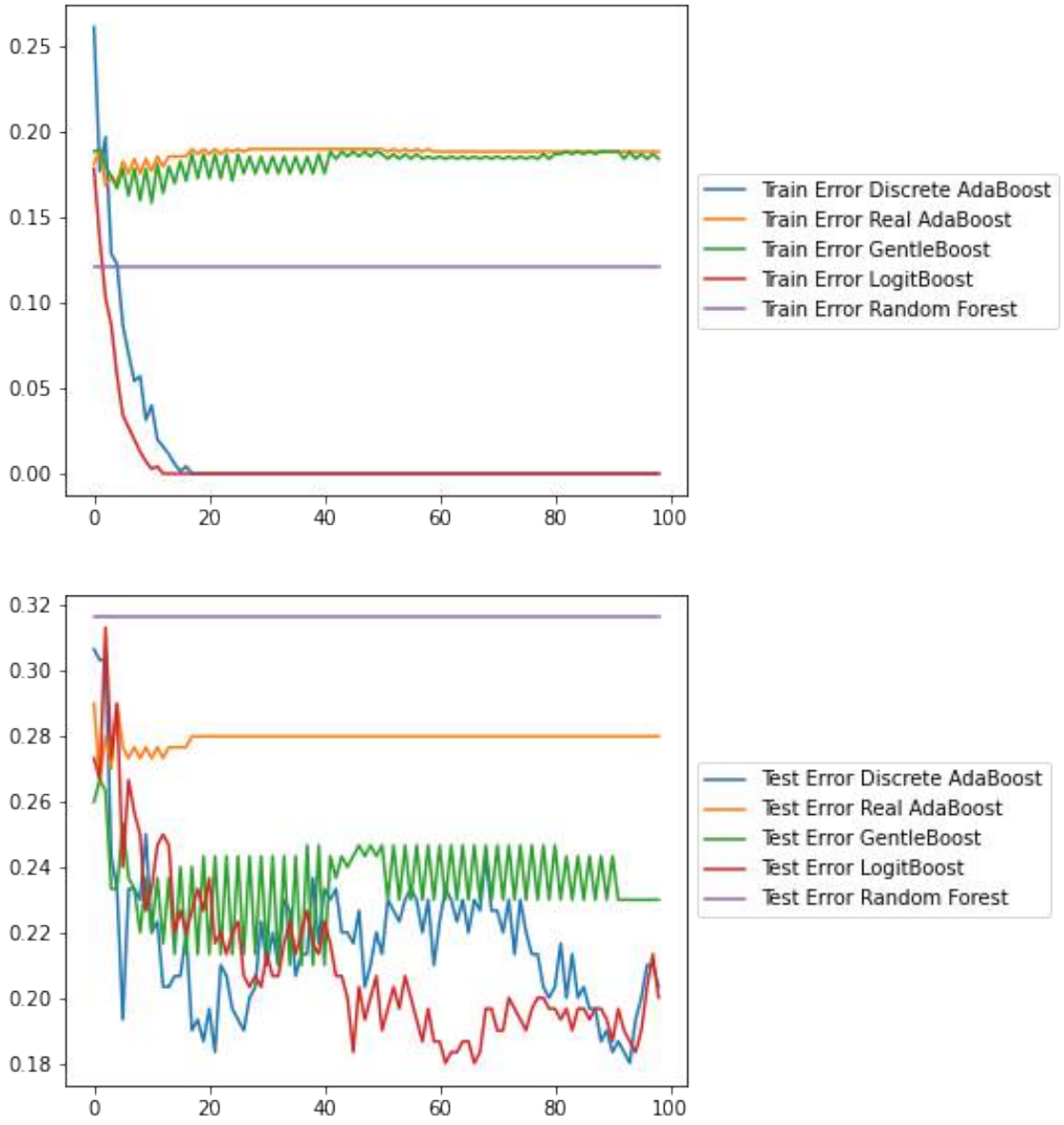

Figura 1.9: Resultados obtenidos al usar árboles de decisión con profundidad máxima igual a 3 sobre datos en dimensión 100

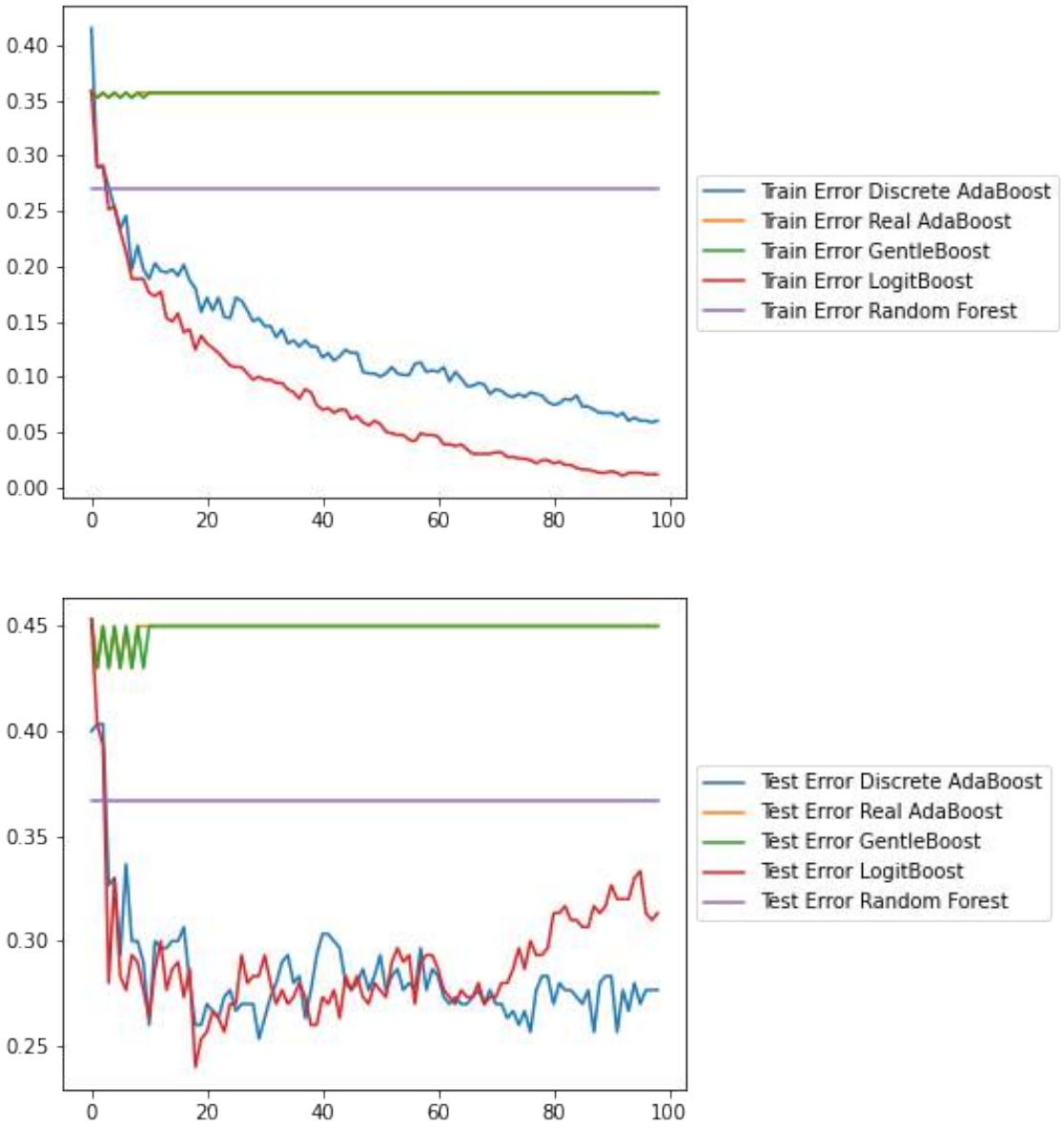

Figura 1.10: Resultados obtenidos al usar árboles de decisión con profundidad máxima igual a 3 sobre datos en dimensión 100

Capítulo 2

Boosting en Regresión Lineal

En el capítulo anterior vimos algunos algoritmos de Boosting que nos permiten optimizar nuestra función objetivo dentro de un espacio de funciones para clasificación. La idea ahora es estudiar algunas aplicaciones de Boosting en regresiones lineales que se desarrollaron en [FGM17]. Por cuestiones de notación y simplicidad, en este caso no vamos a ver los datos como parejas (x_i, y_i), en cambio vamos a escribirlos en forma matricial. De esta forma, si tenemos n datos cada uno con p covariables los vamos a representar en la matriz $\mathbf{X} = [\mathbf{X}_1, ..., \mathbf{X}_p] \in \mathbb{R}^{n \times p}$ y un vector de respuesta $\mathbf{y} \in \mathbb{R}^n$. Vamos a asumir que cada una de las covariables $\mathbf{X}_i$ está centrada de tal forma que su media sea nula y su norma ℓ_2 unitaria. De igual manera, la media del vector de respuesta $\mathbf{y}$ es nula. Cabe aclarar que las suposiciones anteriores no son una restricción de los problemas alcanzados, en cambio, se pueden ver como un pre-procesamiento necesario de los datos. Además, tenemos un vector de coeficientes $\beta \in \mathbb{R}^p$ que usamos para predecir los valores de la regresión calculando $\mathbf{X}\beta$ y $r = \mathbf{y} - \mathbf{X}\beta$ es el residuo. El problema que queremos resolver consiste en encontrar un β que optimice una función objetivo.

Propiedades del problema de mínimos cuadrados

La función que vamos a intentar minimizar es la norma de las diferencia al cuadrado. Veamos algunas propiedades que se tienen sobre esta función de mínimos cuadrados denotada como LS.

$$\text{LS}: \quad L_n^* := \min_{\beta} L_n(\beta) := \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\beta\|_2^2 \qquad (2.1)$$
$$\text{s.a.} \quad \beta \in \mathbb{R}^p,$$

En este caso, $L_n : \mathbb{R}^n \to \mathbb{R}$ es nuestra función de costo. Se está tomando el valor medio de las diferencias entre las predicciones y los valores de $\mathbf{y}$ al cuadrado. Como la función viene dada por una norma entonces los β que solucionan (2.1) cumplen que el gradiente de la función se anula. Es decir, si β^* es una solución de (2.1) entonces

$$\nabla L_n(\beta^*) = -\frac{1}{n}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta^*) = -\frac{1}{n}\mathbf{X}^T r^* = 0. \qquad (2.2)$$

donde $r^* = \mathbf{y} - \mathbf{X}\beta^*$. Además usando (2.2) tenemos que

$$n \cdot \|\nabla L_n(\beta)\|_\infty = \left\|\mathbf{X}^T r\right\|_\infty = \max_{j\in\{1,\ldots,p\}} \left\{\left|r^T \mathbf{X}_j\right|\right\}. \qquad (2.3)$$

El primer algoritmo de Boosting en regresión que vamos a presentar es el de Boosting de mínimos cuadrados(Least Squares Boosting). El algoritmo construye un vector de coeficientes usando una suma de M vectores canónicos multiplicados por un escalar. Para esto el algoritmo actúa de forma *greedy*. Empieza con el vector nulo $\hat{\beta}^0$ y en cada iteración busca el índice j_k de la covariable que mejor disminuye el residuo $r^k = y - X\hat{\beta}^k$. Para encontrar j_k, se calcula en cada covariable la constante $\tilde{u}_m$:

$$\tilde{u}_m = \arg\min_{u\in\mathbb{R}} \left(\sum_{i=1}^{n} (\hat{r}_i^k - x_{im}u)^2\right),$$

y se escoge el índice que minimiza el error cuadrático, en caso de existir varios índices que alcanzan el mínimo se escoge uno al azar.

$$j_k \in \underset{1\leq m\leq p}{\arg\min} \left(\sum_{i=1}^{n} (\hat{r}_i^k - x_{im}\tilde{u}_m)^2 \right),$$

Posteriormente, el algoritmo actualiza la componente j_k del vector de coeficientes utilizando el tamaño del paso (o factor de contracción) $\varepsilon > 0$. De esta manera, tenemos que en cada iteración la actualización del vector de coeficientes varía únicamente en la componente j_k de la siguiente forma: $\hat{\beta}_{j_k}^{k+1} \leftarrow \hat{\beta}_{j_k}^{k} + \varepsilon\tilde{u}_{j_k}$. Nos queda el algoritmo presentado 8.

Un algoritmo muy cercano a LS-Boost(ε) es el algoritmo de Incremental Forward Stagewise Regression (FS_ε), este algoritmo toma el índice j_k de la covariable más relacionada con el residuo $\hat{r}^k$ y actualiza el vector de coeficientes con un paso constante de tamaño ε:

$$\begin{aligned} j_k &\in \underset{j\in\{1,\ldots,p\}}{\arg\max} |(\hat{r}^k)^T \mathbf{X}_j|, \\ \hat{\beta}_{j_k}^{k+1} &\leftarrow \hat{\beta}_{j_k}^{k} + \varepsilon \operatorname{sgn}((\hat{r}^k)^T \mathbf{X}_{j_k}). \end{aligned}$$

Naturalmente podemos hacerle una pequeña modificación al algoritmo de FS_ε para que tome una sucesión de tamaños de paso $\{\varepsilon_k\}$. Con esto las actualizaciones tendrían la siguiente forma:

$$\begin{aligned} j_k &\in \underset{j\in\{1,\ldots,p\}}{\arg\max} |(\hat{r}^k)^T \mathbf{X}_j|, \\ \hat{\beta}_{j_k}^{k+1} &\leftarrow \hat{\beta}_{j_k}^{k} + \varepsilon_k \operatorname{sgn}((\hat{r}^k)^T \mathbf{X}_{j_k}). \end{aligned}$$

a este algoritmo lo llamamos $\text{FS}_{\varepsilon_k}$. Se sigue que podemos ver LS-Boost(ε) se puede ver como un caso especial de $\text{FS}_{\varepsilon_k}$ donde el tamaño del paso ε_k viene dado por $\varepsilon_k = \varepsilon\tilde{u}_{j_k} \operatorname{sgn}((\hat{r}^k)^T X_{j_k})$.

Algoritmo 8 Algoritmo de LS-Boost(ε)

Entrada: Fijamos la tasa de aprendizaje ε.

Inicializar: $\hat{\beta}^0 = 0$

Inicializar: $\hat{r}^0 = \mathbf{y}$

Para k desde 1 hasta M **hacer**

Calcular para cada covariable:

$$\tilde{u}_m = \underset{u \in \mathbb{R}}{\arg\min} \left(\sum_{i=1}^{n} (\hat{r}_i^k - x_{im} u)^2 \right)$$

Encontrar:

$$j_k \in \underset{1 \leq m \leq p}{\arg\min} \left(\sum_{i=1}^{n} (\hat{r}_i^k - x_{im} \tilde{u}_m)^2 \right)$$

Hacer las siguientes actualizaciones:

$$\hat{\beta}_{j_k}^{k+1} \leftarrow \hat{\beta}_{j_k}^k + \varepsilon \tilde{u}_{j_k}$$
$$\hat{r}^{k+1} \leftarrow \hat{r}^k - \varepsilon \mathbf{X}_{j_k} \tilde{u}_{j_k}$$

fin Para

Para constrastar los algoritmos previamente presentados, vamos a tomar en cuenta el problema de Lasso (Least Absolute Shrinkage and Selection Operator). Lasso es un problema de regresión que se caracteriza por realizar regularización de forma explícita y encontrar cuales son las covariables más influyentes. Vamos a usar la siguiente versión de Lasso:

$$\text{Lasso: } L_{n,\delta}^* := \underset{\beta}{\text{mín}} \frac{1}{2n} ||\mathbf{y} - \mathbf{X}\beta||_2^2, \tag{2.4}$$

$$\text{s.a } ||\beta||_1 \leq \delta.$$

En este caso $\delta > 0$ representa el parámetro de regularización. La idea de este es poder robustecer el problema y tener en cuenta únicamente las covariables más importantes. Los algoritmos de Boosting presentados previamente también usan una regularización pero esta se hace de forma implícita por medio de los parámetros ε y M. La regularización presentada en Lasso se hace de forma explícita y se reduce el espacio de soluciones a los vectores en $\mathbb{R}^p$ que tengan norma ℓ_1 sea menor o igual a δ. A continuación vamos a ver propiedades computacionales de los algoritmos de LS-Boost(ε), FS$_\varepsilon$, FS$_{\varepsilon_k}$ y posteriormente un algoritmo que nos permite resolver el problema de Lasso usando métodos de Boosting.

2.1. Propiedades de LS-Boost(ε)

Lema 2.1.1. *Si $\hat{\beta}_{\text{LS}}$ es una solución de* (2.1) *entonces tenemos que*

$$\left\|\mathbf{X}\hat{\beta}^k - \mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2 = 2n\left(L_n(\hat{\beta}^k) - L_n^*\right).$$

Demostración. Para la prueba vamos a utilizar dos propiedades directas de (2.2) que nos dicen que $\langle \mathbf{X}\beta, \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\rangle = 0$ para todo $\beta \in \mathbb{R}^p$ y $L_n^* = \frac{1}{2n}\langle \mathbf{y}, \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\rangle$. De igual manera, tenemos que $\hat{r}^k = \mathbf{y} - \mathbf{X}\hat{\beta}^k$ y podemos incluirlo en el término de la izquierda de la siguiente forma

$$\begin{aligned}
\left\|\mathbf{X}\hat{\beta}^k - \mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2 &= \left\|\mathbf{X}\hat{\beta}^k - \mathbf{y} + \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2 = \left\|-\hat{r}^k + \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2 \\
&= \langle -\hat{r}^k + \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}, -\hat{r}^k + \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\rangle \\
&= \langle \hat{r}^k, \hat{r}^k\rangle - \langle \hat{r}^k, \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\rangle + \langle \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}, -\hat{r}^k + \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\rangle \\
&= \left\|\hat{r}^k\right\|_2^2 - 2\langle \hat{r}^k, \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\rangle + \left\|\mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2 \\
&= \left\|\hat{r}^k\right\|_2^2 - 2\langle \hat{r}^k, \mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\rangle + \left\|\mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2
\end{aligned}$$

$$
\begin{aligned}
&= \left\|\hat{r}^k\right\|_2^2 - 2\langle \mathbf{y} - \mathbf{X}\hat{\beta}^k, \mathbf{y} - \mathbf{X}\hat{\beta}_{\mathrm{LS}}\rangle + \left\|\mathbf{y} - \mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2^2 \\
&= \left\|\hat{r}^k\right\|_2^2 - 2\langle \mathbf{y}, \mathbf{y} - \mathbf{X}\hat{\beta}_{\mathrm{LS}}\rangle - 2\langle \mathbf{X}\hat{\beta}^k, \mathbf{y} - \mathbf{X}\hat{\beta}_{\mathrm{LS}}\rangle + \left\|\mathbf{y} - \mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2^2 \\
&= 2n\left(L_n(\hat{\beta}^k) - 2L_n^* + L_n^*\right) \\
&= 2n\left(L_n(\hat{\beta}^k) - L_n^*\right).
\end{aligned}
$$

□

Teorema 2.1.2. *Consideremos el algoritmo LS-Boost(ε) con tasa de aprendizaje $\varepsilon \in (0, 1]$, definimos $\lambda_{\mathrm{pmin}}(\cdot)$ como el valor propio positivo más pequeño de una matriz y con este la tasa de convergencia lineal γ*

$$
\gamma := \left(1 - \frac{\varepsilon(2-\varepsilon)\lambda_{\mathrm{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}{4p}\right) < 1.
$$

Sea $\hat{\beta}_{\mathrm{LS}}$ una solución de LS (2.1), tenemos que para todo $k > 0$ se cumplen las siguientes cotas:

1. *(Error de entrenamiento): $L_n(\hat{\beta}^k) - L_n^* \leq \frac{1}{2n}\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2^2 \cdot \gamma^k$.*

2. *(Vector de coeficientes) Existe al menos una solución $\hat{\beta}_{LS}^k$ de LS tal que*

$$
\left\|\hat{\beta}^k - \hat{\beta}_{LS}^k\right\|_2 \leq \frac{\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2}{\sqrt{\lambda_{\mathrm{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}} \cdot \gamma^{k/2}.
$$

   Se usa el superíndice k en $\hat{\beta}_{LS}^k$ porque la solución correspondiente puede cambiar entre iteraciones.

3. *(Predicciones) Para toda solución $\hat{\beta}_{LS}$ se cumple*

$$
\left\|\mathbf{X}\hat{\beta}^k - \mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2 \leq \left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2 \cdot \gamma^{k/2}.
$$

4. (Norma del gradiente)

$$\left\|\nabla L_n(\hat{\beta}^k)\right\|_\infty = \frac{1}{n}\left\|\mathbf{X}^T\hat{r}^k\right\|_\infty \leq \frac{1}{n}\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2 \cdot \gamma^{k/2}.$$

5. (Norma ℓ_1 del vector de coeficientes)

$$\left\|\hat{\beta}^k\right\|_1 \leq \min\left\{\sqrt{k}\sqrt{\frac{\varepsilon}{2-\varepsilon}}\sqrt{\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2^2 - \left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}} - \mathbf{X}\hat{\beta}^k\right\|_2^2}, \frac{\varepsilon\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2}{1-\sqrt{\gamma}}\left(1-\gamma^{k/2}\right)\right\}$$

6. (Esparcimiento de los coeficientes):
La cantidad de coeficientes diferentes de 0 es menor o igual a k.

Demostración. Para empezar la prueba, hagamos algunas observaciones sobre las actualizaciones que ocurren en cada iteración del algoritmo. Cada uno de los $\tilde{u}_m$ se puede calcular de forma explícita como $\tilde{u}_m = \sum_{i=1}^n r_i^k x_{im} = (r^k)^T\mathbf{X}_m$. Con esto la actualización del algoritmo se puede reescribir como

$$\hat{r}^{k+1} = \hat{r}^k - \varepsilon\left(\left(\hat{r}^k\right)^T\mathbf{X}_{j_k}\right)\mathbf{X}_{j_k}.$$

Vamos a probar *1.* Tenemos que

$$\begin{aligned}
L_n(\hat{\beta}^{k+1}) &= \frac{1}{2n}\left\|\mathbf{y} - \mathbf{X}\hat{\beta}^{k+1}\right\|_2^2 = \frac{1}{2n}\left\|\hat{r}^{k+1}\right\|_2^2 \\
&= \frac{1}{2n}\left\|\hat{r}^k - \varepsilon\left(\left(\hat{r}^k\right)^T\mathbf{X}_{j_k}\right)\mathbf{X}_{j_k}\right\|_2^2 \\
&= \frac{1}{2n}\left\|\hat{r}^k\right\|_2^2 - \frac{1}{n}\varepsilon\left(\left(\hat{r}^k\right)^T\mathbf{X}_{j_k}\right)^2 + \frac{1}{2n}\varepsilon^2\left(\left(\hat{r}^k\right)^T\mathbf{X}_{j_k}\right) \\
&= L_n(\hat{\beta}^k) - \frac{1}{2n}\varepsilon(2-\varepsilon)\left(\left(\hat{r}^k\right)^T\mathbf{X}_{j_k}\right)^2 \qquad (2.5) \\
&= L_n(\hat{\beta}^k) - \frac{1}{2n}\varepsilon(2-\varepsilon)n^2\left\|\nabla L_n(\hat{\beta}^k)\right\|_\infty^2.
\end{aligned}$$

Restamos L_n^* a ambos lados,

$$L_n(\hat{\beta}^{k+1}) - L_n^* = L_n(\hat{\beta}^k) - L_n^* - \frac{n}{2}\varepsilon(2-\varepsilon)\left\|\nabla L_n(\hat{\beta}^k)\right\|_\infty^2. \tag{2.6}$$

La idea es usar el teorema (A.1.2) usando que L_n es una función cuadrática convexa, para esto la reescribimos como

$$L_n(\beta) = \frac{1}{2n}\left\|\mathbf{y} - \mathbf{X}\hat{\beta}\right\|_2^2 = \frac{1}{2}\beta^T\left(\frac{1}{n}\mathbf{X}^T\mathbf{X}\right)\beta - \frac{1}{2n}\left(\mathbf{y}^T\mathbf{X}\right)\beta + \frac{1}{2n}\|\mathbf{y}\|_2^2.$$

Por el teorema tenemos que

$$\left\|\nabla L_n(\hat{\beta})\right\|_2 \geq \sqrt{\frac{\lambda_{\text{pmin}}\left(\frac{1}{n}\mathbf{X}^T\mathbf{X}\right)\left(L_n(\hat{\beta}) - L_n^*\right)}{2}} = \sqrt{\frac{\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)\left(L_n(\hat{\beta}) - L_n^*\right)}{2n}}.$$

Y por lo tanto

$$\|\nabla L_n(\beta)\|_\infty^2 \geq \frac{1}{p}\|\nabla L_n(\beta)\|_2^2 \geq \frac{\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)\left(L_n(\beta) - L_n^*\right)}{2np}, \tag{2.7}$$

usando la desigualdad (2.7) en (2.6) nos queda que

$$L_n(\hat{\beta}^{k+1}) - L_n^* \leq \left(L_n(\hat{\beta}^k) - L_n^*\right)\left(1 - \frac{\varepsilon(2-\varepsilon)\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}{4p}\right) = \left(L_n(\hat{\beta}^k) - L_n^*\right)\cdot\gamma.$$

Con esto tenemos una forma recursiva de relacionar $\left(L_n(\hat{\beta}^k) - L_n^*\right)$ y $\left(L_n(\hat{\beta}^{k+1}) - L_n^*\right)$. Nos falta calcular el caso base para obtener una desigualdad general. Tenemos que $L_n(\hat{\beta}^0) = L_n(0) = \frac{1}{2n}\|\mathbf{y}\|_2^2$ y de (2.2) tenemos que si $\hat{\beta}_{\text{LS}}$ es una solución de LS entonces $\langle y, \mathbf{X}\hat{\beta}_{\text{LS}}\rangle = \left\|\mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2$. Se sigue que

$$\begin{aligned}
L_n\left(\hat{\beta}^0\right) - L_n^* &= \frac{1}{2n}\|\mathbf{y}\|_2^2 - \frac{1}{2n}\left\|\mathbf{y} - \mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2 \\
&= \frac{1}{2n}\|\mathbf{y}\|_2^2 - \frac{1}{2n}\left(\|\mathbf{y}\|_2^2 - 2\mathbf{y}^T\mathbf{X}\hat{\beta}_{\text{LS}} + \left\|\mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2\right) \\
&= \frac{1}{2n}\|\mathbf{y}\|_2^2 - \frac{1}{2n}\left(\|\mathbf{y}\|_2^2 - \left\|\mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2\right)
\end{aligned}$$

$$= \frac{1}{2n} \left\| \mathbf{X}\hat{\beta}_{\mathrm{LS}} \right\|_2^2 .$$

Por último, usando el principio de inducción podemos concluir que

$$L_n\left(\hat{\beta}^k\right) - L_n^* \leq \left(L_n\left(\hat{\beta}^0\right) - L_n^*\right) \cdot \gamma^k = \frac{1}{2n} \left\| \mathbf{X}\hat{\beta}_{\mathrm{LS}} \right\|_2^2 \cdot \gamma^k. \tag{2.8}$$

Para probar *2* vamos a usar la primera desigualdad del teorema (A.1.2) y el punto anterior (2.8). La desigualdad nos dice que

$$\left\| \hat{\beta}^k - \hat{\beta}_{\mathrm{LS}} \right\|_2 \leq \frac{\sqrt{2\left(L_n(\hat{\beta}^k) - L_n^*\right)}}{\sqrt{\lambda_{\mathrm{pmin}}\left(\frac{1}{n}\mathbf{X}^T\mathbf{X}\right)}} = \frac{\sqrt{2n\left(L_n(\hat{\beta}^k) - L_n^*\right)}}{\sqrt{\lambda_{\mathrm{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}},$$

Se sigue que

$$\left\| \hat{\beta}^k - \hat{\beta}_{\mathrm{LS}} \right\|_2 \leq \frac{\sqrt{\left\| \mathbf{X}\hat{\beta}_{\mathrm{LS}} \right\|_2^2 \cdot \gamma^k}}{\sqrt{\lambda_{\mathrm{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}} = \frac{\left\| \mathbf{X}\hat{\beta}_{\mathrm{LS}} \right\|_2}{\sqrt{\lambda_{\mathrm{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}} \cdot \gamma^{k/2},$$

Al igual que en el punto anterior, para probar *3* usamos el lema (2.1.1) y la desigualdad (2.8). Nos queda

$$\begin{aligned}
\left\| \mathbf{X}\hat{\beta}^k - \mathbf{X}\hat{\beta}_{\mathrm{LS}} \right\|_2 &= \sqrt{2n\left(L_n(\hat{\beta}^k) - L_n^*\right)} \\
&\leq \sqrt{2n \cdot \frac{1}{2n} \left\| \mathbf{X}\hat{\beta}_{\mathrm{LS}} \right\|_2^2 \cdot \gamma^k} \\
&\leq \left\| \mathbf{X}\hat{\beta}_{\mathrm{LS}} \right\|_2 \cdot \gamma^{k/2}.
\end{aligned}$$

Para probar el punto *4*, usamos el vector $\tilde{\beta}^k := \hat{\beta}^k + \tilde{u}_{j_k} e_{j_k}$. Usando la definición de L_n^* y que $\tilde{u}_{j_k} = (\hat{r}^k)^T \mathbf{X}_{j_k}$ tenemos que

$$L_n^* \leq L_n(\tilde{\beta}^k)$$

$$
\begin{aligned}
&\leq= L_n(\hat{\beta}^k) - \frac{1}{n}(\hat{r}^k)^T \mathbf{X}_{j_k} \tilde{u}_{j_k} + \frac{1}{2n} \left\| \tilde{u}_{j_k} e_{j_k} \right\|_2^2 \\
&\leq L_n(\hat{\beta}^k) - \frac{1}{2n} \tilde{u}_{j_k}^2 .
\end{aligned}
$$

Usando (2.3) y (2.8) nos queda que

$$
\begin{aligned}
\tilde{u}_{j_k}^2 &\leq 2n \left(L_n(\hat{\beta}^k) - L_n^* \right) \\
\left\| \mathbf{X}^T (\hat{r}^k) \right\|_\infty = \left| \tilde{u}_{j_k} \right| &\leq \sqrt{2n \left(L_n(\hat{\beta}^k) - L_n^* \right)} \\
n \cdot \| \nabla L_n(\beta) \|_\infty = \left\| \mathbf{X}^T (\hat{r}^k) \right\|_\infty &\leq \left\| \mathbf{X} \hat{\beta}_{\mathrm{LS}} \right\|_2 \cdot \gamma^{k/2} \\
\| \nabla L_n(\beta) \|_\infty = \frac{1}{n} \left\| \mathbf{X}^T (\hat{r}^k) \right\|_\infty &\leq \frac{1}{n} \left\| \mathbf{X} \hat{\beta}_{\mathrm{LS}} \right\|_2 \cdot \gamma^{k/2} .
\end{aligned}
\tag{2.9}
$$

La propiedad *5* se divide en dos desigualdades. Para la primera, vamos a usar (2.1.1) y (2.5). Tenemos para $i = 0, \ldots, k-1$ que

$$
\begin{aligned}
\left\| \mathbf{X} \hat{\beta}^i - \mathbf{X} \hat{\beta}_{\mathrm{LS}} \right\|_2^2 &= 2n \left(L_n(\hat{\beta}^i) - L_n^* \right), \\
\left\| \mathbf{X} \hat{\beta}^{i+1} - \mathbf{X} \hat{\beta}_{\mathrm{LS}} \right\|_2^2 &= 2n \left(L_n(\hat{\beta}^{i+1}) - L_n^* \right).
\end{aligned}
$$

Entonces,

$$
\begin{aligned}
\left\| \mathbf{X} \hat{\beta}^i - \mathbf{X} \hat{\beta}_{\mathrm{LS}} \right\|_2^2 - \left\| \mathbf{X} \hat{\beta}^{i+1} - \mathbf{X} \hat{\beta}_{\mathrm{LS}} \right\|_2^2 &= 2n \left(L_n(\hat{\beta}^i) - L_n(\hat{\beta}^{i+1}) \right) \\
&= \varepsilon(2 - \varepsilon) \left(\left(\hat{r}^k \right)^T \mathbf{X}_{j_k} \right)^2 \\
&= \varepsilon(2 - \varepsilon) \tilde{u}_{j_k}^2 .
\end{aligned}
$$

Se sigue que

$$
\left(2\varepsilon - \varepsilon^2 \right) \sum_{i=0}^{k-1} \tilde{u}_{j_i}^2 = \left\| \mathbf{X}(\hat{\beta}^0 - \hat{\beta}_{\mathrm{LS}}) \right\|_2^2 - \left\| \mathbf{X}(\hat{\beta}^k - \hat{\beta}_{\mathrm{LS}}) \right\|_2^2 = \left\| \mathbf{X} \hat{\beta}_{\mathrm{LS}} \right\|_2^2 - \left\| \mathbf{X}(\hat{\beta}^k - \hat{\beta}_{\mathrm{LS}}) \right\|_2^2 .
$$

Por lo tanto, con la desigualdad de Cauchy-Schwartz nos queda

$$\begin{aligned}\left\|\hat{\beta}^k\right\|_1 &\leq \left\|\left(\varepsilon\tilde{u}_{j_0},\ldots,\varepsilon\tilde{u}_{j_{k-1}}\right)\right\|_1 \\ &\leq \sqrt{k}\varepsilon\left\|\left(\tilde{u}_{j_0},\ldots,\tilde{u}_{j_{k-1}}\right)\right\|_2 \\ &= \sqrt{k}\sqrt{\frac{\varepsilon}{2-\varepsilon}}\sqrt{\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2^2 - \left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}} - \mathbf{X}\hat{\beta}^k\right\|_2^2}.\end{aligned}$$

Para la segunda desigualdad, vamos a usar (2.9). Nos queda

$$\begin{aligned}\left\|\hat{\beta}^k\right\|_1 &\leq \varepsilon\sum_{i=0}^{k-1}\left|\tilde{u}_{j_i}\right| \\ &\leq \varepsilon\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2\sum_{i=0}^{k-1}\gamma^{i/2} \\ &\leq \varepsilon\left\|\mathbf{X}\hat{\beta}_{LS}\right\|_2\frac{\left(1-\gamma^{k/2}\right)}{1-\sqrt{\gamma}}\left(1-\gamma^{k/2}\right).\end{aligned}$$

Por último, la propiedad *6* viene del hecho que $\hat{\beta}^0 := 0$ y en cada iteración a lo sumo una coordenada deja de ser nula. □

La mayoría de propiedades del teorema anterior están relacionadas con el valor γ, este se define como la tasa de convergencia lineal. Si definimos $\kappa\left(\mathbf{X}^T\mathbf{X}\right) := \frac{p}{\lambda_{\mathrm{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}$ entonces podemos escribir $\gamma = 1-\frac{\varepsilon(2-\varepsilon)}{4\kappa\left(\mathbf{X}^T\mathbf{X}\right)}$. Para entender mejor los valores que puede tomar γ tomamos el vector propio β_λ asociado al valor propio más grande de $\mathbf{X}^T\mathbf{X}$ y usamos que $\|\mathbf{X}\|_{1,2} = \max\limits_{\beta:\|\beta\|_1\leq 1}\|\mathbf{X}\beta\|_2 = \max\left(\|\mathbf{X}_1\|_2,\ldots,\left\|\mathbf{X}_p\right\|_2\right)$. De esta forma, nos queda que

$$0 < \lambda_{\mathrm{pmin}}\left(\mathbf{X}^T\mathbf{X}\right) \leq \lambda_{\mathrm{máx}}\left(\mathbf{X}^T\mathbf{X}\right) = \frac{\|\mathbf{X}\beta_\lambda\|_2^2}{\|\beta_\lambda\|_2^2} \leq \frac{\|\mathbf{X}\|_{1,2}^2\|\beta_\lambda\|_1^2}{\|\beta_\lambda\|_2^2} \leq p,$$

tenemos que $\|\mathbf{X}\|_{1,2}^2 = 1$ porque todas las columnas de $\mathbf{X}$ están normalizadas y por la desigualdad de Cauchy-Schwartz $\|\beta_\lambda\|_1 \leq \sqrt{p}\|\beta_\lambda\|_2$. Por lo tanto, nos queda que

$\kappa\left(\mathbf{X}^T\mathbf{X}\right) \in [1,\infty)$ y $\gamma \in [0{,}75, 1{,}0)$. Es importante notar que $\gamma < 1$ sin importar cual sea el conjunto de datos que estamos usando.

El teorema anterior no permite encontrar garantías computacionales del algoritmo antes de correrlo. Podemos calcular explícitamente el parámetro γ y escoger el k para que se cumpla la cota deseada. La primera converge linealmente junto con el parámetro γ para *1* y para los puntos *2* a *4* con el parámetro $\sqrt{\gamma}$. Por lo anterior, las cotas tienden a 0 cuando $k \to \infty$. Otro punto importante del algoritmo anterior es que no necesita que la matriz $\mathbf{X}^T\mathbf{X}$ sea definida positiva. Si el valor propio más pequeño es nulo, de igual manera se tiene una convergencia global. Esto se debe a que los valores de la función de mínimos cuadrados $L_n(\cdot)$ son invariantes dentro del espacio nulo de $\mathbf{X}$. En la siguiente sección cambiaremos la perspectiva con la que nos estamos apróximando a los problemas y vamos a tener en cuenta los residuos.

2.2. Boosting como descenso de subgradiente

El propósito de esta sección es ver los algoritmos previamente presentados como diferentes instancias del método de descenso subgradiente para el problema de minimizar la norma infinita de la correlación entre los residuos y los predictores(datos utilizados para predecir). La idea principal es cambiar totalmente la perspectiva con la cual se está aproximando al problema. A partir de ahora vamos a pensar principalmente en los residuos.
Para esto, definimos $P_{\text{res}} := \{r \in \mathbb{R}^n : r = \mathbf{y} - \mathbf{X}\beta \text{ para algún} \beta \in \mathbb{R}^p\}$ que denota el espacio afín asociado a los residuos. En el caso especial que P_{res} sea un espacio lineal, esto implica que existe un $\beta \in \mathbb{R}^p$ tal que $\mathbf{y} = \mathbf{X}\beta$. Consideremos el siguiente problema de optimización:

$$\text{Mínima Correlación (MC)} : \quad f^* := \min_r f(r) := \left\|\mathbf{X}^T r\right\|_\infty \tag{2.10}$$

$$\text{s.t. } r \in P_{\text{res}} \, .$$

Algo importante a tener en cuenta es que en este problema la variable a optimizar es el residuo r a diferencia de los otros problemas en los que optimizábamos el vector de coeficientes β.
Recordemos que las columnas de $\mathbf{X}$ tienen norma ℓ_2 unitaria, por lo tanto, $f(r)$ representa la mayor correlación entre el vector de residuos y y los predictores. Al utilizar (2.2) con $r = \mathbf{y} - \mathbf{X}\beta$, podemos ver que β es una solución de LS si y solo si $\mathbf{X}^T r = 0$, por lo que $f(r) = \left\|\mathbf{X}^T r\right\|_\infty = 0$ para el vector de residuos r implica que es el vector de residuos de una solución del problema (2.1) y $r = \hat{r}_{LS} = \mathbf{y} - \mathbf{X}\hat{\beta}_{LS}$. Dado que $f(r) \geq 0$ para todo $r \in P_{\text{res}}$ entonces se sigue que $f^* = 0$ y $\hat{r}_{LS}$ es una solución de MC. Con lo anterior, tenemos que el problema de MC es un problema de optimización que soluciona el problema de mínimos cuadrados.

Proposición 2.2.1. *Si consideramos el algoritmo de descenso de subgradiente presentado en* (A.2) *con los tamaños de los pasos dados por la secuencia* $\{\alpha_k\}$ *para resolver el problema de mínima correlación (MC)* (2.10)*, empezando con* $\hat{r}^0 = \mathbf{y}$*. Entonces:*

a) el algoritmo de FS_ε *es una instancia de descenso de subgradiente con pasos constantes* $\alpha_k := \varepsilon$ *en cada iteración.*

b) el algoritmo de $\text{FS}_{\varepsilon_k}$ *es una instancia de descenso de subgradiente con pasos de* $\alpha_k := \varepsilon_k$ *en la* k*-ésima iteración.*

c) el algoritmo de LS-Boost(ε) *es una instancia de descenso de subgradiente con tamaños de paso de* $\alpha_k := \varepsilon \left|\tilde{u}_{jk}\right|$ *en la* k*-ésima iteración.*

Demostración. Empecemos por probar *a)*, tenemos que en cada iteración de FS_ε las actualizaciones de los residuos vienen dadas por:

$$\hat{r}^{k+1} = \hat{r}^k - \varepsilon \cdot \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right) \mathbf{X}_{j_k}.$$

La idea es mostrar que $\mathrm{g}^k := \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k}$ es un subgradiente de la función objetivo $f(r) = \left\|\mathbf{X}^T r\right\|_\infty$ en el punto $r = \hat{r}^k$. En la k-ésima iteración, el algoritmo de FS_ε actualiza la componente $j_k \in \arg\max_{j\in\{1,\ldots,p\}} \left|\left(\hat{r}^k\right)^T \mathbf{X}_j\right|$ y $\operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right) = \left\|\mathbf{X}^T\left(\hat{r}^k\right)\right\|_\infty$. Se sigue que para todo $r \in P_{\mathrm{res}}$ se cumple que

$$\begin{aligned}
f(r) &\geq \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\left(\left(\mathbf{X}_{j_k}\right)^T r\right) \\
&= \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\left(\left(\mathbf{X}_{j_k}\right)^T\left(\hat{r}^k + r - \hat{r}^k\right)\right) \\
&= \left\|\mathbf{X}^T\left(\hat{r}^k\right)\right\|_\infty + \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\left(\left(\mathbf{X}_{j_k}\right)^T\left(r - \hat{r}^k\right)\right) \\
&= f\left(\hat{r}^k\right) + \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\left(\left(\mathbf{X}_{j_k}\right)^T\left(r - \hat{r}^k\right)\right) \\
&= f\left(\hat{r}^k\right) + \left(\operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\left(\mathbf{X}_{j_k}\right)\right)^T\left(r - \hat{r}^k\right).
\end{aligned}$$

Se sigue de la definición de subgradiente en (A.1.4) que g^k es un subgradiente de $f(r)$ en el punto $r = \hat{r}^k$. De esta forma, la actualización $\hat{r}^{k+1} = \hat{r}^k - \varepsilon \cdot \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k}$ se puede ver cómo $\hat{r}^{k+1} = \hat{r}^k - \varepsilon \mathrm{g}^k$ con $\mathrm{g}^k \in \partial f\left(\hat{r}^k\right)$. Por último, veamos que $\hat{r}^k - \varepsilon \mathrm{g}^k \in P_{\mathrm{res}}$:

$$\begin{aligned}
\hat{r}^{k+1} &= \hat{r}^k - \varepsilon \mathrm{g}^k \\
&= \mathbf{y} - \mathbf{X}\hat{\beta}^k - \varepsilon \mathrm{g}^k \\
&= \mathbf{y} - \mathbf{X}\hat{\beta}^k - \varepsilon \cdot \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k} \\
&= \mathbf{y} - \mathbf{X}\left(\hat{\beta}^k - \varepsilon \cdot \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right) e_{j_k}\right) \\
&= \mathbf{y} - \mathbf{X}(\hat{\beta}^{k+1}).
\end{aligned}$$

tenemos que $\hat{\beta}^{k+1} \in \mathbb{R}^p$ y por lo tanto $\Pi_{P_{\mathrm{res}}}\left(\hat{r}^{k+1}\right) = \hat{r}^{k+1} \in P_{\mathrm{res}}$.

La prueba de *b)* es análoga a la prueba de *a)* tomando el tamaño del paso en la

k-ésima iteración como $\alpha_k = \varepsilon_k$. De igual manera, podemos ver LS-Boost(ε) como un caso especial de $\text{FS}_{\varepsilon_k}$ donde $\varepsilon_k := \varepsilon \tilde{u}_{jk} \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)$. □

La proposición anterior nos permite ver a los tres algoritmos FS_ε, $\text{FS}_{\varepsilon_k}$ y LS-Boost(ε) se pueden ver como instancias del método de subgradiente para resolver el problema MC (2.10). Ahora la idea es ver algunas propiedades computacionales que se siguen de la interpretación de FS_ε como una instancia del método de descenso de subgradiente.

Teorema 2.2.1. *Consideremos el algoritmo de* FS_ε *con taza de aprendizaje* ε*. Sea* $k \geq 0$ *el número total de iteraciones, entonces existe al menos un* $i \in \{0, \ldots, k\}$ *para el cual se cumplen las siguientes cotas:*

1. *(Error de entrenamiento):* $L_n(\hat{\beta}^i) - L_n^* \leq \frac{p}{2n\lambda_{\text{pmin}}(\mathbf{X}^T\mathbf{X})} \left[\frac{\|\mathbf{X}\hat{\beta}_{\text{LS}}\|_2^2}{\varepsilon(k+1)} + \varepsilon\right]^2$.

2. *(Vector de coeficientes) Existe al menos una solución* $\hat{\beta}^i_{LS}$ *de LS tal que*

$$\left\|\hat{\beta}^i - \hat{\beta}^i_{LS}\right\|_2 \leq \frac{\sqrt{p}}{\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)} \left[\frac{\left\|\mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2}{\varepsilon(k+1)} + \varepsilon\right].$$

3. *(Predicciones) Para toda solución* $\hat{\beta}_{LS}$ *se cumple*

$$\left\|\mathbf{X}\hat{\beta}^i - \mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2 \leq \frac{\sqrt{p}}{\sqrt{\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}} \left[\frac{\left\|\mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2}{\varepsilon(k+1)} + \varepsilon\right].$$

4. *(Correlación)*

$$\left\|\mathbf{X}^T\hat{r}^i\right\|_\infty \leq \frac{\left\|\mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2}{2\varepsilon(k+1)} + \frac{\varepsilon}{2}.$$

5. *(Norma* ℓ_1 *del vector de coeficientes)*

$$\left\|\hat{\beta}^i\right\|_1 \leq k\varepsilon.$$

6. *(Esparcimiento de los coeficientes):*
   La cantidad de coeficientes diferentes de 0 es menor o igual a k.

Demostración. La prueba de este teorema es muy similar a la prueba de 2.1.2, se utiliza la primera cota para demostrar el resto. Por lo tanto solo vamos a probar *1.* Tenemos que FS_ε es una instancia de descenso de subgradiente con tamaño de paso igual a ε. Por lo tanto, podemos usar el teorema A,3 dentro del contexto de resolver el problema MC 2.10 con FS_ε. Usando (2.2) obtenemos que $f^* = 0$. Además, podemos escribir la distancia entre los residuos como

$$\left\|\hat{r}^0 - r^*\right\|_2 = \left\|\hat{r}^0 - \hat{r}_{LS}\right\|_2 = \left\|\mathbf{y} - \left(\mathbf{y} - \mathbf{X}\hat{\beta}_{\mathrm{LS}}\right)\right\|_2 = \left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2 .$$

Teniendo en cuenta que el tamaño del paso α en este caso es igual a ε,entonces, para poder usar A,3 solo nos falta encontrar una cota superior G para las normas de los subgradientes. Tenemos que

$$\left\|g^k\right\|_2 = \left\|\mathrm{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k}\right\|_2 = \left\|\mathbf{X}_{j_k}\right\|_2 = 1.$$

Dado que las covariables tienen norma ℓ_2 unitaria tenemos que $G = 1$ corresponde al supremo de las normas de los subgradientes. Si suponemos que el algoritmo lleva k iteraciones se sigue que

$$\min_{i\in\{0,\ldots,k\}} \left\|\mathbf{X}^T\hat{r}^i\right\|_\infty = \min_{i\in\{0,\ldots,k\}} f\left(\hat{r}^i\right) \leq f^* + \frac{\left\|\hat{r}^0 - r^*\right\|_2^2}{2\alpha(k+1)} + \frac{\alpha G^2}{2} = \frac{\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2^2}{2\varepsilon(k+1)} + \frac{\varepsilon}{2} \quad (2.11)$$

Usando (2.3) nos queda que

$$\min_{i\in\{0,\ldots,k\}} \left\|\nabla L_n\left(\hat{\beta}^i\right)\right\|_\infty \leq \frac{\left\|\mathbf{X}\hat{\beta}_{\mathrm{LS}}\right\|_2^2}{2n\varepsilon(k+1)} + \frac{\varepsilon}{2n} \quad (2.12)$$

Notemos que la última desigualdad es sobre el conjunto de $\left\{\hat{\beta}^i\right\}$ y se puede controlar el número de iteraciones y la tasa de aprendizaje ε. Sin pérdida de generalidad

asumamos que i corresponde al índice donde la norma $\nabla L_n(\cdot)$ alcanza su mínimo. Previamente en (2.7) habíamos calculado que

$$\left\|\nabla L_n(\hat{\beta}^i)\right\|_\infty^2 \geq \frac{1}{p}\left\|\nabla L_n(\hat{\beta}^i)\right\|_2^2 \geq \frac{\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)\left(L_n(\hat{\beta}^i) - L_n^*\right)}{2np}.$$

Para terminar de probar *1* reemplazamos $\left\|\nabla L_n(\hat{\beta}^i)\right\|_\infty^2$ en (2.12) y reorganizamos:

$$\begin{aligned}
\sqrt{\frac{\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)\left(L_n(\hat{\beta}^i) - L_n^*\right)}{2np}} &\leq \frac{\left\|\mathbf{X}\hat{\beta}^i_{\text{LS}}\right\|_2^2}{2n\varepsilon(k+1)} + \frac{\varepsilon}{2n} \\
\frac{\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)\left(L_n(\hat{\beta}^i) - L_n^*\right)}{2np} &\leq \left[\frac{\left\|\mathbf{X}\hat{\beta}^i_{\text{LS}}\right\|_2^2}{2n\varepsilon(k+1)} + \frac{\varepsilon}{2n}\right]^2 \\
L_n(\hat{\beta}^i) - L_n^* &\leq \frac{p}{2n\lambda_{\text{pmin}}\left(\mathbf{X}^T\mathbf{X}\right)}\left[\frac{\left\|\mathbf{X}\hat{\beta}^i_{\text{LS}}\right\|_2^2}{\varepsilon(k+1)} + \varepsilon\right]^2.
\end{aligned}$$

Como lo dijimos previamente, la prueba de los otros puntos es muy similar a la prueba de 2.1.2 usando la cota del punto *1*. □

Si lo comparamos con el teorema 2.1.2 vemos que la convergencia de FS_ε es sublineal a diferencia de la convergencia lineal de LS-Boost(ε). Esto se debe a las actualizaciones de los residuos en cada iteración de los algoritmos:

$$\begin{aligned}
\text{LS-Boost}(\varepsilon): \quad & \left\|\hat{r}^{k+1} - \hat{r}^k\right\|_2 = \varepsilon\left|\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right| = \varepsilon \cdot n \cdot \left\|\nabla L_n\left(\hat{\beta}^k\right)\right\|_\infty, \\
\text{FS}_\varepsilon: \quad & \left\|\hat{r}^{k+1} - \hat{r}^k\right\|_2 = \varepsilon\,|s_k| \quad \text{donde } s_k = \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right).
\end{aligned}$$

En cada iteración el algoritmo de FS_ε toma pasos de tamaño ε a menos que llegue al óptimo, en cambio, LS-Boost(ε) puede dar pasos muy grandes en las primeras iteraciones y por lo tanto converge mucho más rápido. Además, también existe una

diferencia en la precisión de las soluciones. Cuando $k \to \infty$ tenemos que LS-Boost(ε) alcanza una solución global pero FS_ε converge con una precisión de $O(\varepsilon)$.

2.2.1. Boosting y el problema de Mínima Correlación Regularizado

Sea $\delta \in (0, \infty]$ el parámetro de regularización para el problema de mínima correlación 2.10 entonces podemos definir el problema de mínima correlación regularizado (MCR_δ) como

$$\text{MCR}_\delta : \quad f_\delta^* := \min_r f_\delta(r) := \left\|\mathbf{X}^T r\right\|_\infty + \frac{1}{2\delta}\|r - \mathbf{y}\|_2^2, \qquad (2.13)$$
$$\text{s.a.} \quad r \in P_{\text{res}} \, .$$

Lema 2.2.2. *La función de pérdida de mínimos cuadrados $L_n(\cdot)$ tiene la siguiente representación como máximo:*

$$L_n(\beta) = \max_{\tilde{r} \in P_{res}} \left\{ -\tilde{r}^T \left(\frac{1}{n}\mathbf{X}\right)\beta - \frac{1}{2n}\|\tilde{r} - \mathbf{y}\|_2^2 + \frac{1}{2n}\|\mathbf{y}\|_2^2 \right\}. \qquad (2.14)$$

Demostración. Para empezar, encontremos el $r \in P_{\text{res}}$ tal que

$$r = \arg\max_{\tilde{r} \in P_{\text{res}}} \left\{ -\tilde{r}^T \left(\frac{1}{n}\mathbf{X}\right)\beta - \frac{1}{2n}\|\tilde{r} - \mathbf{y}\|_2^2 + \frac{1}{2n}\|\mathbf{y}\|_2^2 \right\}.$$

Para esto, calculamos el gradiente de (2.14) con respecto a $\tilde{r}$ y lo igualamos a cero.

$$\nabla L_n(\beta) = -\frac{1}{n}\mathbf{X}\beta - \frac{1}{n}(\tilde{r} - \mathbf{y}) = 0,$$
$$\to \quad \tilde{r} = \mathbf{y} - \mathbf{X}\beta.$$

Al remplazar $\tilde{r} = \mathbf{y} - \mathbf{X}\beta$ en (2.14) nos queda:

$$\max_{\tilde{r} \in P_{\text{res}}} \left\{ -\tilde{r}^T \left(\frac{1}{n}\mathbf{X}\right)\beta - \frac{1}{2n}\|\tilde{r} - \mathbf{y}\|_2^2 + \frac{1}{2n}\|\mathbf{y}\|_2^2 \right\} = -(\mathbf{y} - \mathbf{X}\beta)^T \left(\frac{1}{n}\mathbf{X}\right)\beta - \frac{1}{2n}\|\mathbf{X}\beta\|_2^2 + \frac{1}{2n}\|\mathbf{y}\|_2^2$$

$$= \frac{1}{n}\left(-\mathbf{y}^T\mathbf{X}\beta + \|\mathbf{X}\beta\|_2^2 - \frac{1}{2}\|\mathbf{X}\beta\|_2^2 + \frac{1}{2}\|\mathbf{y}\|_2^2\right),$$
$$= \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\beta\|_2^2.$$

□

Proposición 2.2.2 (Equivalencia de dualidad entre Lasso y MCR_δ)**.** *El problema de Lasso 2.4 y el problema de mínima correlación regularizado* MCR_δ *2.13 son problemas duales modulo el factor* $\frac{n}{\delta}$*. Por consecuente, se cumple:*

1. *(Dualidad Débil) Si* β *es factible para el problema Lasso, y si* $\tilde{r}$ *es factible para el problema de mínima correlación regularizado* MCR_δ*, entonces*

$$L_n(\beta) + \frac{\delta}{n} f_\delta(\tilde{r}) \geq \frac{1}{2n}\|\mathbf{y}\|_2^2.$$

2. *(Dualidad Fuerte) Se tiene que*

$$L_{n,\delta}^* + \frac{\delta}{n} f_\delta^* = \frac{1}{2n}\|\mathbf{y}\|_2^2.$$

3. *(Condición de optimalidad para Lasso) Si* β *es factible para el problema Lasso y* $r = \mathbf{y} - \mathbf{X}\beta$*, entonces*

$$\omega_\delta(\beta) := \left\|\mathbf{X}^T r\right\|_\infty - \frac{r^T\mathbf{X}\beta}{\delta} \geq 0, \tag{2.15}$$

y

$$L_n(\beta) - L_{n,\delta}^* \leq \frac{\delta}{n} \cdot \omega_\delta(\beta).$$

Por lo tanto, si $\omega_\delta(\beta) = 0$ *entonces* β *is solución óptima del problema de Lasso.*

Demostración. Para probar la proposición en una primera instancia construyamos el problema MCR_δ 2.13 usando la representación de $L_n(\cdot)$ presentada en (2.14):

$$L_n(\beta) = \max_{\tilde{r} \in P_{\text{res}}} \left\{ -\tilde{r}^T \left(\frac{1}{n} \mathbf{X} \right) \beta - \frac{1}{2n} \|\tilde{r} - \mathbf{y}\|_2^2 + \frac{1}{2n} \|\mathbf{y}\|_2^2 \right\}.$$

Si definimos $B_\delta := \{\beta \in \mathbb{R}^p : \|\beta\|_1 \leq \delta\}$ entonces podemos escribir el problema de Lasso como:

$$\min_{\beta \in B_\delta} \max_{\tilde{r} \in P_{\text{res}}} \left\{ -\tilde{r}^T \left(\frac{1}{n} \mathbf{X} \right) \beta - \frac{1}{2n} \|\tilde{r} - \mathbf{y}\|_2^2 + \frac{1}{2n} \|\mathbf{y}\|_2^2 \right\}.$$

Construimos el siguiente problema dual intercambiando los operadores de mín y máx:

$$\max_{\tilde{r} \in P_{\text{res}}} \min_{\beta \in B_\delta} \left\{ -\tilde{r}^T \left(\frac{1}{n} \mathbf{X} \right) \beta - \frac{1}{2n} \|\tilde{r} - \mathbf{y}\|_2^2 + \frac{1}{2n} \|\mathbf{y}\|_2^2 \right\}.$$

Si descartamos el término constante $\frac{1}{2n}\|\mathbf{y}\|_2^2$, el problema dual anterior es equivalente a

$$\min_{\tilde{r} \in P_{\text{res}}} \max_{\beta \in B_\delta} \left\{ \tilde{r}^T \left(\frac{1}{n} \mathbf{X} \right) \beta \right\} + \frac{1}{2n} \|\tilde{r} - \mathbf{y}\|_2^2. \tag{2.16}$$

Ahora veamos que

$$\max_{\beta \in B_\delta} \left\{ \tilde{r}^T \left(\frac{1}{n} \mathbf{X} \right) \beta \right\} = \frac{\delta}{n} \left(\max_{j \in \{1,\ldots,p\}} \left| \tilde{r}^T \mathbf{X}_j \right| \right) = \frac{\delta}{n} \left\| \mathbf{X}^T \tilde{r} \right\|_\infty.$$

Lo anterior se debe a que se está maximizando sobre los $\beta \in B_\delta$ entonces máximo se alcanza cuando β toma el valor de $\pm\delta$ en la componente con mayor valor absoluto. Después de multiplicar por $\frac{n}{\delta}$ se sigue que (2.16) es equivalente a 2.13.

Ahora podemos probar el punto *1*. Sean β factible para el problema Lasso y $\tilde{r}$ factible para el problema de mínima correlación regularizado MCR_δ. Definimos r y $\tilde{\beta}$ tales que $r = \mathbf{y} - \mathbf{X}\beta$ y $\tilde{r} = \mathbf{y} - \mathbf{X}\tilde{\beta}$. Entonces se sigue que

$$\begin{aligned}
L_n(\beta) + \frac{\delta}{n} f_\delta(\tilde{r}) &= \frac{1}{2n} ||\mathbf{y} - \mathbf{X}\beta||_2^2 + \frac{\delta}{n} \left(\left\| \mathbf{X}^T \tilde{r} \right\|_\infty + \frac{1}{2\delta} \|\tilde{r} - \mathbf{y}\|_2^2 \right) \\
&= \frac{1}{2n} ||r||_2^2 + \frac{\delta}{n} \left\| \mathbf{X}^T \tilde{r} \right\|_\infty + \frac{1}{2n} \left(\|\tilde{r}\|_2^2 + \|\mathbf{y}\|_2^2 - 2\langle r, \mathbf{y} \rangle \right) \\
&= \frac{1}{2n} ||r||_2^2 + \frac{\delta}{n} \left\| \mathbf{X}^T \tilde{r} \right\|_\infty + \frac{1}{2n} \left(\|\tilde{r}\|_2^2 + \|\mathbf{y}\|_2^2 - 2\langle \tilde{r}, r + \mathbf{X}\beta \rangle \right)
\end{aligned}$$

$$= \frac{1}{2n}\|\mathbf{y}\|_2^2 + \frac{1}{2n}\left(||r||_2^2 - 2\langle\tilde{r}, r\rangle + \|\tilde{r}\|_2^2\right) + \frac{\delta}{n}\left\|\mathbf{X}^T\tilde{r}\right\|_\infty - \frac{1}{n}\langle\tilde{r}, \mathbf{X}\beta\rangle$$
$$= \frac{1}{2n}\|\mathbf{y}\|_2^2 + \frac{1}{2n}\|r - \tilde{r}\|_2^2 + \frac{\delta}{n}\left(\left\|\mathbf{X}^T\tilde{r}\right\|_\infty - \frac{\tilde{r}^T\mathbf{X}\beta}{\delta}\right) \tag{2.17}$$

Notemos que con la desigualdad de Hölder tenemos que $\tilde{r}^T\mathbf{X}\beta \leq \left\|\mathbf{X}^T\tilde{r}\right\|_\infty \|\beta\|_1 \leq \delta\left\|\mathbf{X}^T\tilde{r}\right\|_\infty$. Por lo tanto, los términos $\|r - \tilde{r}\|_2^2$ y $\frac{1}{\delta}\left(\delta\left\|\mathbf{X}^T\tilde{r}\right\|_\infty - \tilde{r}^T\mathbf{X}\beta\right)$ siempre son mayores o iguales a cero y se tiene la desigualdad deseada. El punto *2* se sigue directamente del hecho que Lasso y MCR_δ son problemas de optimización con una función objetivo convexa y cuadrática con restricciones de desigualdades lineales. Para el punto *3*, tomemos β factible para el problema de Lasso y al igual que para *1* la desigualdad de Hölder nos dice que $\omega_\delta(\beta) \geq 0$. Usamos el resultado de (2.17) con $\tilde{r} \leftarrow r = \mathbf{y} - \mathbf{X}\beta$ tenemos que:

$$L_n(\beta) + \frac{\delta}{n}f_\delta(r) = \frac{1}{2n}\|\mathbf{y}\|_2^2 + \frac{\delta}{n}\cdot\omega_\delta(\beta)$$

Usando también el punto *2* nos queda:

$$L_n(\beta) + \frac{\delta}{n}f_\delta(r) = L_{n,\delta}^* + \frac{\delta}{n}f_\delta^* + \frac{\delta}{n}\cdot\omega_\delta(\beta).$$

La última desigualdad viene de la definición de f_δ^* que implica que $f_\delta^* \leq f_\delta(r)$.

□

Con el fin de encontrar una solución para el problema MCR 2.13 vamos a realizar unos cambios al algoritmo de FS_ε de tal forma que la norma ell_1 del vector de coeficientes no supere el parámetro de regularización δ. Con esto, obtenemos el algoritmo de $\text{R} - \text{FS}_{\varepsilon,\delta}$:

Algoritmo 9 Algoritmo de $\mathrm{R}-\mathrm{FS}_{\varepsilon,\delta}$

Entrada: Fijar la taza de aprendizaje $\varepsilon > 0$, el parámetro de regularización $\delta > \varepsilon$ y el número de iteraciones M.

Inicializar: Sea $\hat{\beta}^0 = 0$.

Para k desde 1 hasta M **hacer**

Tomar $j_k \in \underset{j\in\{1,\ldots,p\}}{\arg\max} \left|\left(\hat{r}^k\right)^T \mathbf{X}_j\right|$.

Hacer las siguientes actualizaciones:

$$\begin{aligned}
\hat{r}^{k+1} &\leftarrow \hat{r}^k - \varepsilon\left[\operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k} + \frac{1}{\delta}\left(\hat{r}^k - \mathbf{y}\right)\right],\\
\hat{\beta}_{j_k}^{k+1} &\leftarrow \left(1 - \frac{\varepsilon}{\delta}\right)\hat{\beta}_{jk}^k + \varepsilon \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right),\\
\hat{\beta}_j^{k+1} &\leftarrow \left(1 - \frac{\varepsilon}{\delta}\right)\hat{\beta}_j^k, \quad \text{para } j \neq j_k
\end{aligned}$$

fin Para

Al igual que en la sección anterior, veremos que el algoritmo de $\mathrm{R}-\mathrm{FS}_{\varepsilon,\delta}$ se puede ver como una instancia de descenso de subgradiente y cuales son las garantías computacionales obtenidas.

Proposición 2.2.3. *El algoritmo de* $\mathrm{R}-\mathrm{FS}_{\varepsilon,\delta}$ *se puede ver como una instancia de descenso de subgradiente para resolver el problema de mínima correlación regularizado* (2.13)*, inicializado en* $\hat{r}^0 = \mathbf{y}$ *con un paso constante de tamaño* $\alpha_k := \varepsilon$ *para cada iteración* k*.*

Demostración. Tenemos que la actualización de los residuos en cada iteración viene dada por:

$$\hat{r}^{k+1} \leftarrow \hat{r}^k - \varepsilon\left[\operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k} + \frac{1}{\delta}\left(\hat{r}^k - \mathbf{y}\right)\right].$$

Veamos que $g^k := \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k} + \frac{1}{\delta}\left(\hat{r}^k - \mathbf{y}\right)$ es un subgradiente de $f_\delta(\cdot)$ en el punto $\hat{r}^k$. Usando la prueba de 2.2.1 tenemos que $\operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k}$ es un sub-

gradiente de $f(r) := \left\|\mathbf{X}^T r\right\|_\infty$ en el punto $\hat{r}^k$ y con $j_k \in \arg\max_{j\in\{1,\dots,p\}} \left|\left(\hat{r}^k\right)^T \mathbf{X}_j\right|$. Ahora, notemos que $f_\delta(r) = f(r) + \frac{1}{2\delta}\|r-\mathbf{y}\|_2^2$. Se sigue por la propiedad de aditividad de los subgradientes (y gradientes) que

$$g^k = \operatorname{sgn}\left(\left(\hat{r}^k\right)^T \mathbf{X}_{j_k}\right)\mathbf{X}_{j_k} + \frac{1}{\delta}\left(\hat{r}^k - \mathbf{y}\right),$$

es un subgradiente de $f_\delta(r)$ en el punto $r = \hat{r}^k$. Por lo tanto la actualización del algoritmo es de la forma $\hat{r}^{k+1} = \hat{r}^k - \varepsilon g^k$ donde $g^k \in \partial f_\delta\left(\hat{r}^k\right)$. Por último, dado que $\hat{r}^k - \varepsilon g^k = \hat{r}^{k+1} = \mathbf{y} - \mathbf{X}\beta^{k+1} \in P_{\text{res}}$ entonces $\Pi_{P_{\text{res}}}\left(\hat{r}^k - \varepsilon g^k\right) = \hat{r}^k - \varepsilon g^k$. Lo que muestra que la actualización del algoritmo de $\mathrm{R}-\mathrm{FS}_{\varepsilon,\delta}$ es una instancia del método de descenso de gradiente con paso de tamaño ε. □

Teorema 2.2.3. *Consideremos el algoritmo de* $\mathrm{R}-\mathrm{FS}_{\varepsilon,\delta}$ *con tasa de aprendizaje de* ε *y parámetro de regularización* $\delta \in (0,\infty)$ *con* $\delta \geq \epsilon$. *Entonces el vector de coeficientes* $\hat{\beta}^k$ *es factible para el problema de Lasso 2.4 para todo* $k \geq 0$ *y existe algún* $i \in \{0,\dots,k\}$ *para el cual se cumplen las siguientes cotas:*

1. *(Error de entrenamiento):* $L_n\left(\hat{\beta}^i\right) - L_{n,\delta}^* \leq \frac{\delta}{n}\left[\frac{\left\|\mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2}{2\varepsilon(k+1)} + 2\varepsilon\right]$;

2. *(Predicciones) Para toda solución* $\hat{\beta}_\delta^*$ *de Lasso se cumple que*

$$\left\|\mathbf{X}\hat{\beta}^i - \mathbf{X}\hat{\beta}_\delta^*\right\|_2 \leq \sqrt{\frac{\delta\left\|\mathbf{X}\hat{\beta}_{\text{LS}}\right\|_2^2}{\varepsilon(k+1)} + 4\delta\varepsilon};$$

3. *(Norma* ℓ_1 *del vector de coeficientes)* $\left\|\hat{\beta}^i\right\|_1 \leq \delta\left[1 - \left(1 - \frac{\varepsilon}{\delta}\right)^k\right] \leq \delta$;

4. *(Esparcimiento de los coeficientes):*
   La cantidad de coeficientes diferentes de 0 es menor o igual a k.

Al igual que para LS-Boost(ε) y FS_ε el algoritmo nos muestra garantías que se tienen para $\mathrm{R}-\mathrm{FS}_{\varepsilon,\delta}$ con respecto a las soluciones óptimas de Lasso. Podemos ver que cuado $k \to \infty$, el resultado de $\mathrm{R}-\mathrm{FS}_{\varepsilon,\delta}$ aproxima la solución de Lasso con una

precisión de $O(\varepsilon)$.

2.3. Resultados

En esta sección se mostrarán los resultados obtenidos al probar los algoritmos de regresión. Los comparamos con las soluciones óptimas de LS 2.1 y Lasso 2.4. Para empezar, tomamos datos en dimensión 10 y se tiene que hay 4 covariables que cargan la mayor parte de la información sobre el valor de cada dato.

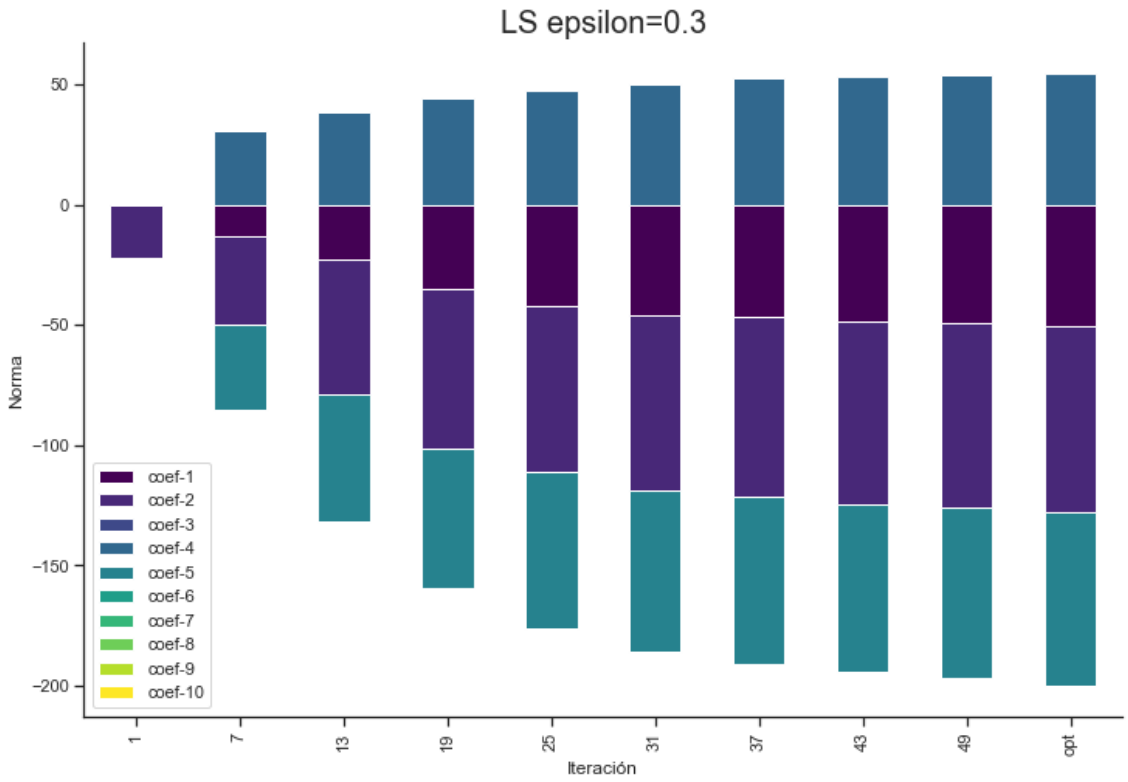

Figura 2.1: Norma y valores del vector de coeficientes en diferentes iteraciones de LS-Boost(0.3), la última barra representa el valor óptimo de los coeficientes de mínimos cuadrados.

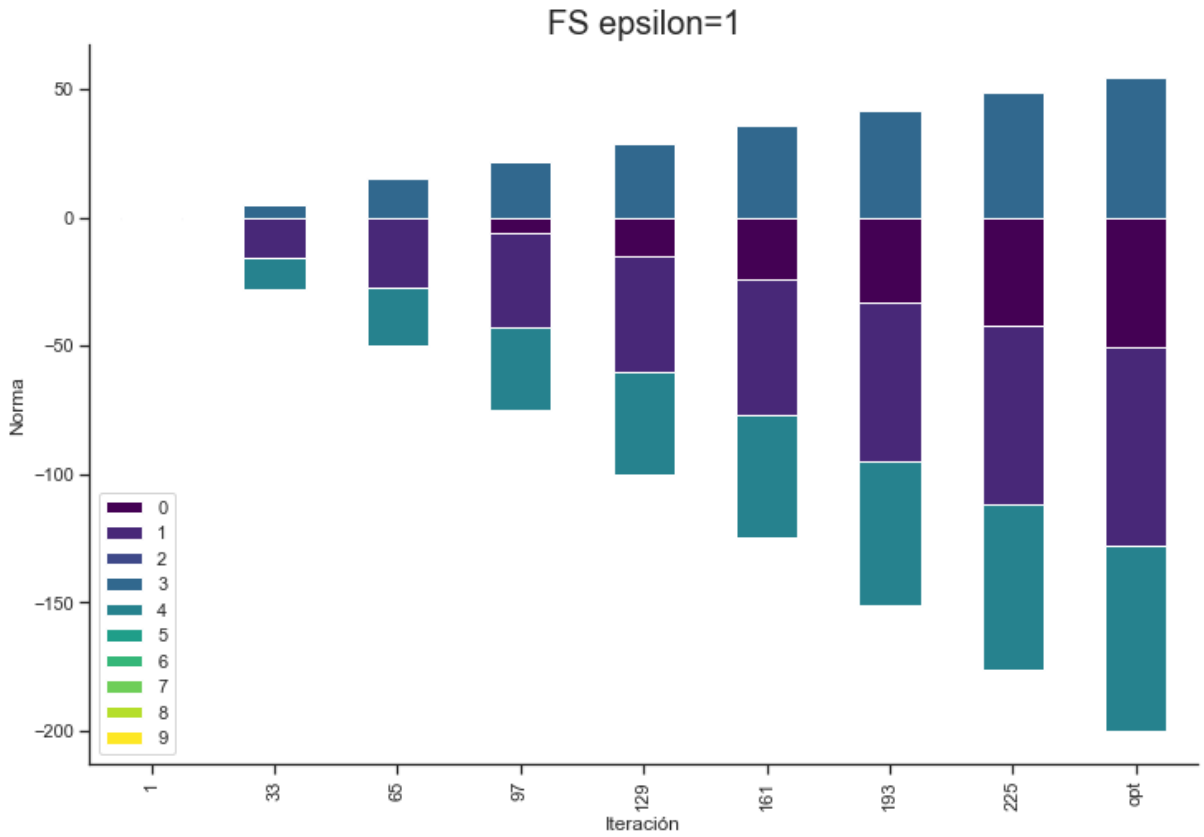

Figura 2.2: Norma y valores del vector de coeficientes en diferentes iteraciones de FS_1, la última barra representa el valor óptimo de los coeficientes de mínimos cuadrados.

En 2.1 y 2.2 podemos ver las diferencias de los algoritmos de LS-Boost($\boldsymbol{\varepsilon}$) y FS_{ε} en términos de su velocidad de convergencia. El algoritmo de LS-Boost(0.3) tiene un tamaño de paso menor, solo se corren 50 iteraciones y se acerca bastante a la solución óptima. En cambio, para FS_1 se corren 225 iteraciones y tiene un rendimiento mucho menor. Esto es acorde con lo que se probó en la sección anterior.

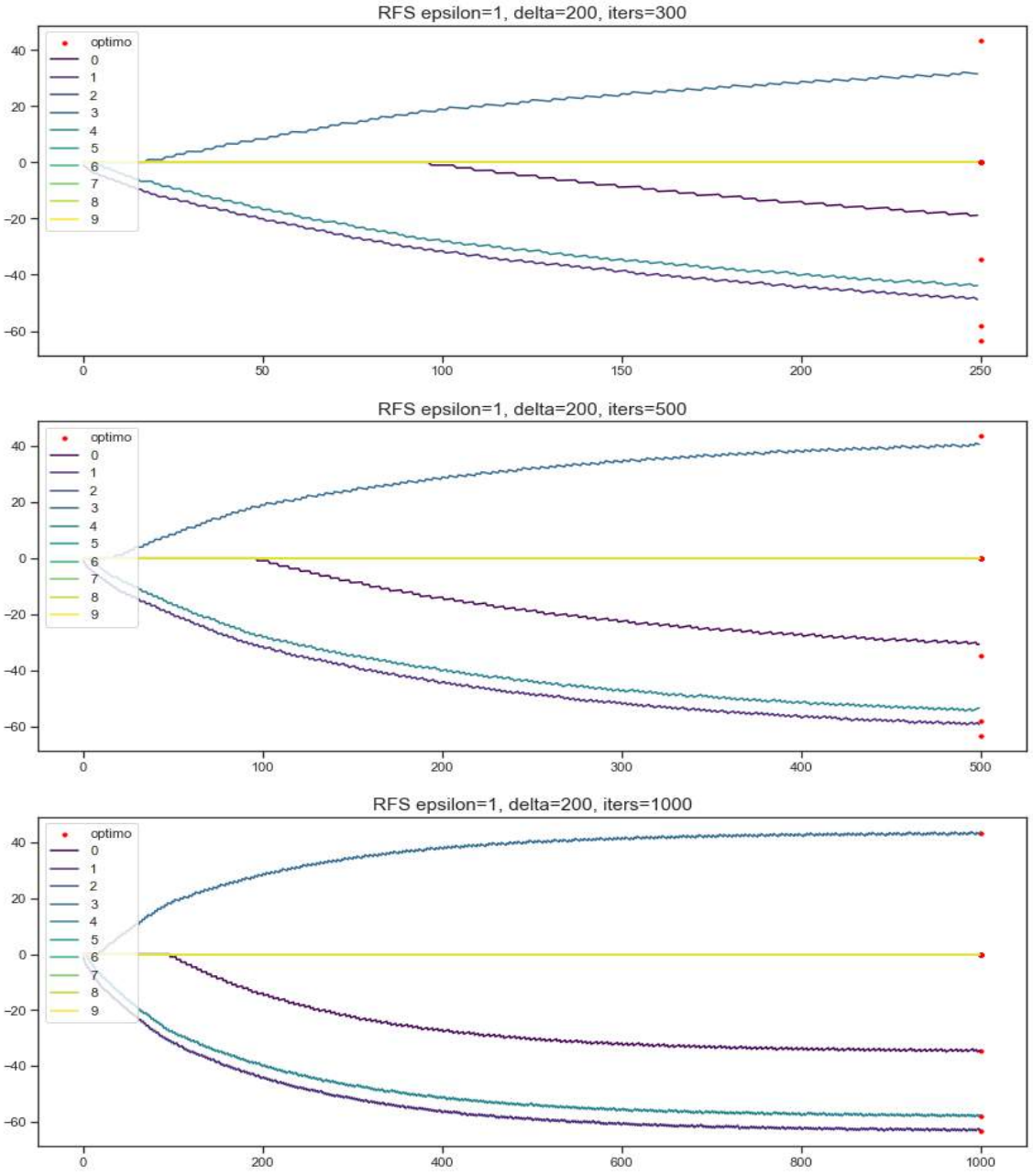

Figura 2.3: Valores de los vector de coeficientes en tres corridas de RFS_1 para $\delta = 200$ y 250,500 y 1000 iteraciones respectivamente. Los puntos representan los valores óptimos de los coeficientes de Lasso para $\delta = 200$.

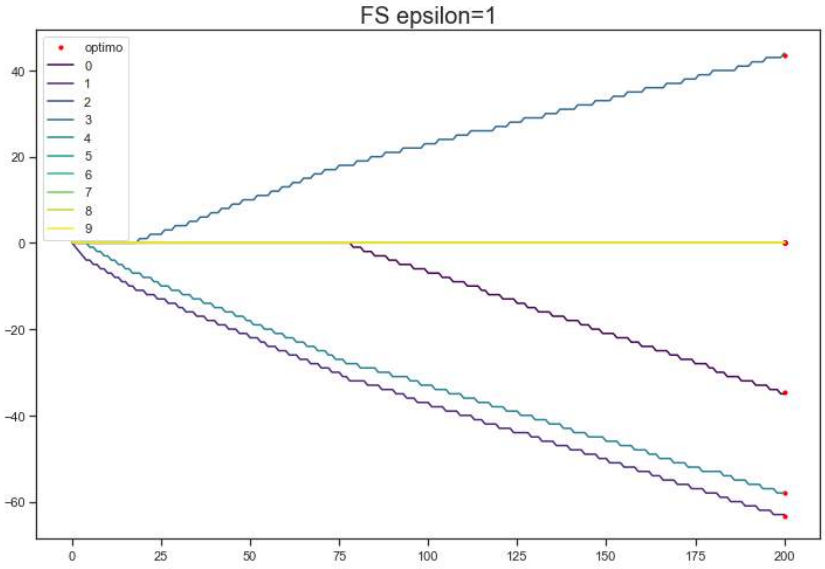

Figura 2.4: Valores del vector de coeficientes de FS_1 a lo largo de las 200 iteraciones, los puntos rojos representan los valores óptimo de los coeficientes de Lasso.

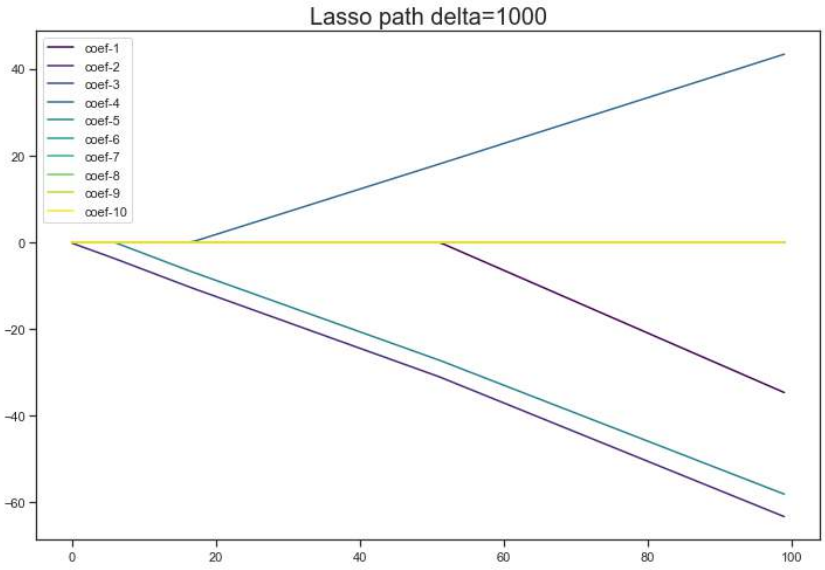

Figura 2.5: Valores del vector de coeficientes de Lasso para diferentes valores de δ. Este se conoce como el Lasso path.

La idea de 2.3 y 2.4 es comparar la regulación implícita del algoritmo FS_ε y la regularización dada por el parámetro $\delta = 200$ en Lasso y $\mathrm{R}-\mathrm{FS}_{\varepsilon,\delta}$. Con los valores de $\varepsilon = 1$ y el máximo de iteraciones igual a 200, tenemos que la norma del vector de coeficientes de FS_1 es menor o igual a 200. Sin embargo, si aumentamos el número de iteraciones, el resultado de FS_1 dejaría de ser factible para el problema de Lasso. Por otro lado, vemos que el algoritmo de $\mathrm{R}-\mathrm{FS}_{1,200}$ siempre tiene una norma ℓ_1 menor que δ.

Una de las principales ventajas de estos algoritmos es que alcanzan las soluciones óptimas con muy buena precisión y el tiempo computacional aumenta de forma moderada respecto a la cantidad y dimensión de los datos. En cambio, para encontrar las soluciones exactas hay que invertir una matriz y esto puede llevar a problemas computacionales. Por esto, se hicieron pruebas para datos en dimensión 100 y se obtuvieron los siguientes resultados:

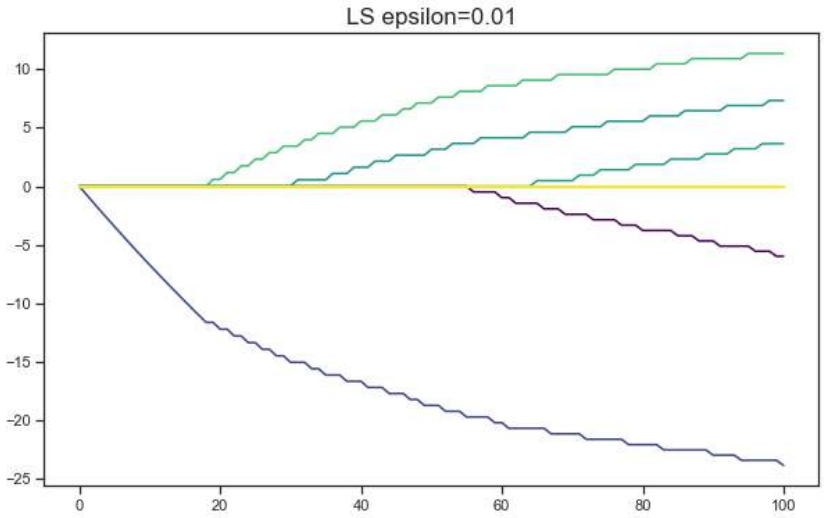

Figura 2.6: Valores del vector de coeficientes de LS-Boost para $\varepsilon = 0{,}01$ a lo largo de 100 iteraciones.

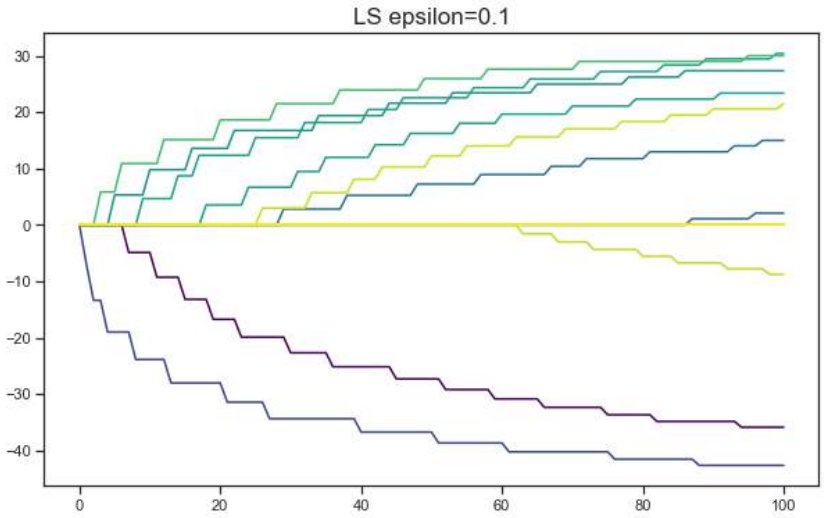

Figura 2.7: Valores del vector de coeficientes de LS-Boost para $\varepsilon = 0{,}1$ a lo largo de 100 iteraciones.

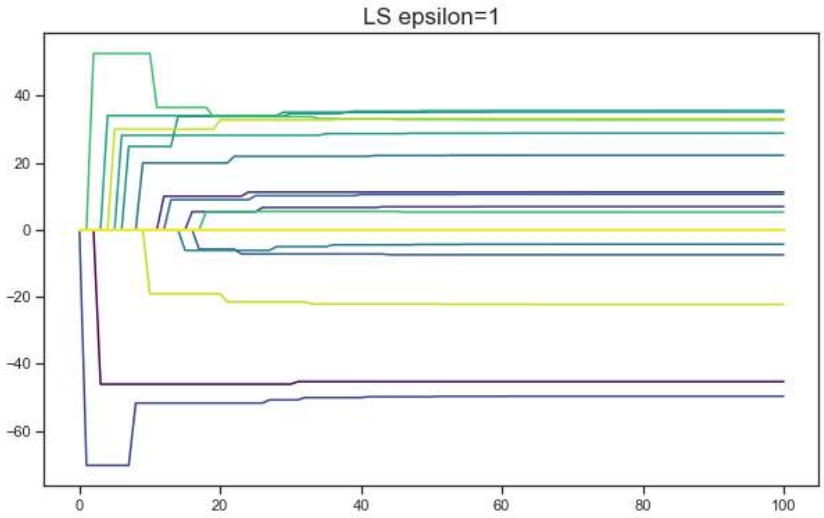

Figura 2.8: Valores del vector de coeficientes de LS-Boost para $\varepsilon = 1$ a lo largo de 100 iteraciones.

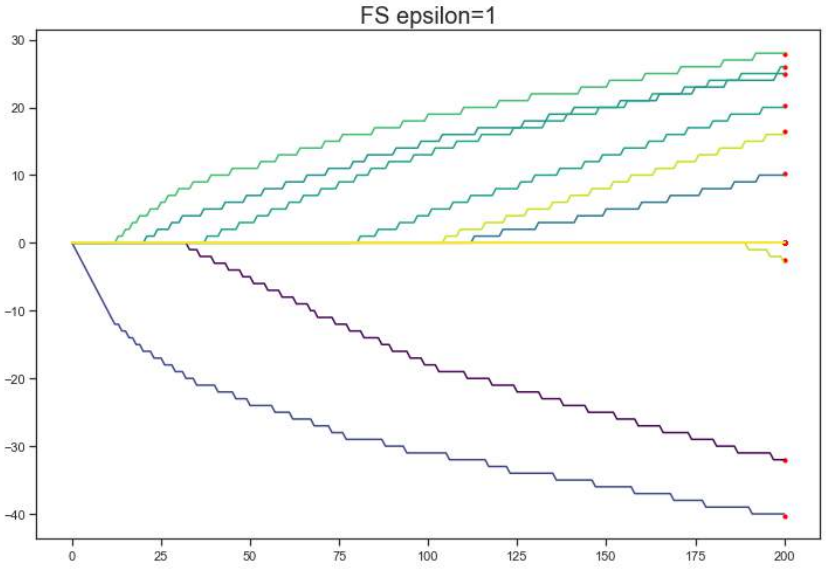

Figura 2.9: Valores del vector de coeficientes de FS_ε para $\varepsilon = 1$ a lo largo de 200 iteraciones.

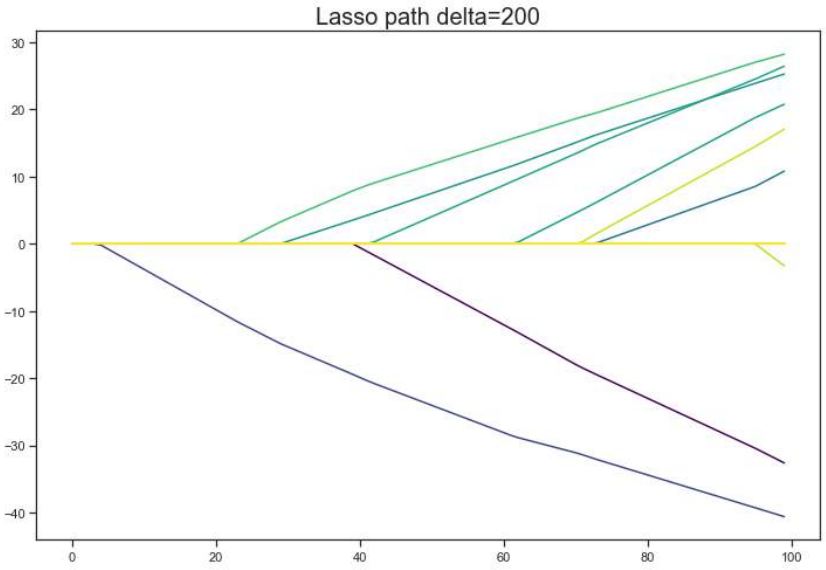

Figura 2.10: Valores del vector de coeficientes de Lasso para diferentes valores de δ. Este se conoce como el Lasso path.

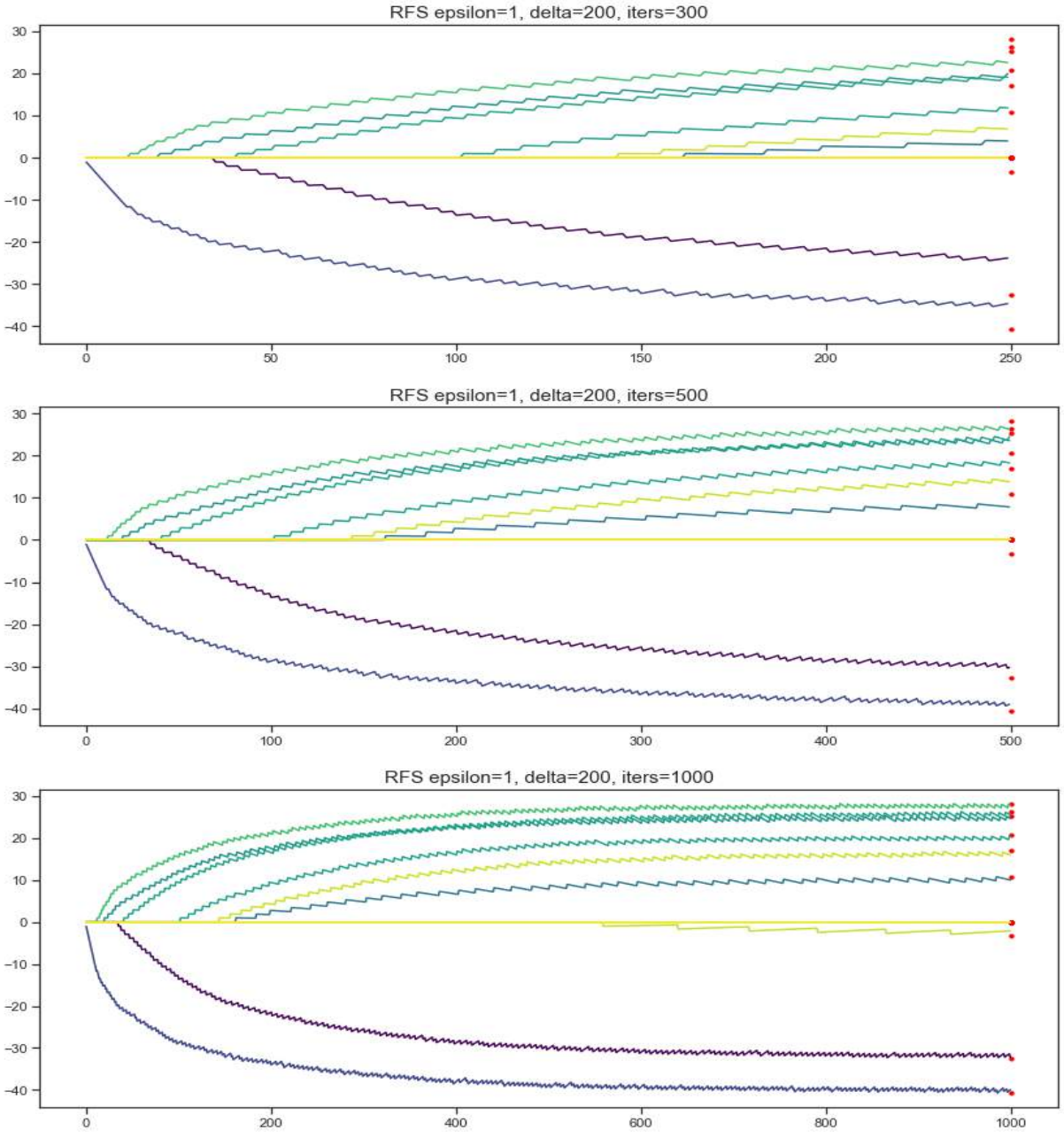

Figura 2.11: Valores de los vector de coeficientes en tres corridas de RFS_1 para $\delta = 200$ y 250,500 y 1000 iteraciones respectivamente. Los puntos representan los valores óptimos de los coeficientes de Lasso para $\delta = 200$.

Apéndice A

Contenido adicional

A.1. Definiciones, teoremas y métodos

En este apéndice se añaden algunas definiciones y teoremas que se utilizaron en el presente trabajo y pueden permitir un mejor entendimiento de este. Las definiciones y teoremas expuestos tienen como referencia principal [BV04]

Definición A.1.1. *Definimos un problema de optimización cuadrático (QP) como*

$$h^* := \min_{x \in \mathbb{R}^n} h(x) := \frac{1}{2} x^T Q x + q^T x + q^o,$$

donde Q es una matriz simétrica semidefinida positiva y por lo tanto $h(\cdot)$ es una función convexa.

Teorema A.1.2. *Supongamos que tenemos un problema de optimización cuadrático (QP) tal que $Q \neq 0$. Denotamos $\lambda_{pmin}(Q)$ como el valor propio positivo más pequeño de Q. Se cumple que si $h^* > -\infty$, entonces para todo x existe una solución x^* de (QP) para la cual:*

$$\|x - x^*\|_2 \leq \sqrt{\frac{2\,(h(x) - h^*)}{\lambda_{\text{pmin}}(Q)}}.$$

Además, se tiene que

$$\|\nabla h(x)\|_2 \geq \sqrt{\frac{\lambda_{\text{pmin}}(Q) \cdot (h(x) - h^*)}{2}}.$$

Para las siguientes definiciones y teoremas, consideramos el siguiente problema de optimización:

$$f^* := \min_x f(x) \tag{A.1}$$

$$\text{s.a. } x \in X \subset \mathbb{R}^n,$$

donde X es cerrado y convexo y $f : X \to \mathbb{R}$ es una función convexa.

Consideremos que $f(\cdot)$ es una función diferenciable, entonces $f(\cdot)$ satisface la siguiente desigualdad para todo $x, y \in X$

$$f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

Un método intuitivo para resolver (A.1) es el método de descenso de gradiente: Dado un punto inicial $x^0 \in X$. Si x^k es la iteración actual, se define la siguiente actualización:

$$x^{k+1} \leftarrow \Pi_X \left(x^k - \alpha_k \nabla f(x^k) \right).$$

Tenemos que $\alpha_k > 0$ representa el tamaño del paso en la iteración k y $\Pi_X(\cdot)$ es el operador de proyección sobre X que nos permite mantener la solución dentro del dominio de $f(\cdot)$. Ahora veamos que pasa si la función $f(\cdot)$ no es diferenciable.

Definición A.1.3 (Subgradiente)**.** *Se dice que g es un subgradiente de una función f en el punto $x \in X$ si se cumple la siguiente desigualdad:*

$$f(y) \geq f(x) + g^T (y - x) \quad \textit{para todo } y \in X,$$

esto generaliza el concepto de gradiente para cualquier función $f(\cdot)$.

Definición A.1.4 (Subdiferencial)**.** *Definimos el subdiferencial de $f(\cdot)$ en x como el conjunto de todos los subgradientes de $f(\cdot)$ en x y lo denotamos como $\partial f(x)$.*

Cabe notar que si $f(\cdot)$ es una función convexa entonces para cada $x \in X$ tiene por lo menos un subgradiente. Con esto, podemos explicar el método de descenso de subgradiente. El método de subgradiente es una generalización del método de descenso de gradiente para cuando la función $f(\cdot)$ no es diferenciable. Para este, en cada iteración calculamos algún subgradiente $g^k \in \partial f(x^k)$ y la actualización viene dada por:

$$x^{k+1} \leftarrow \Pi_X \left(x^k - \alpha_k g^k \right). \tag{A.2}$$

Teorema A.1.5. *Consideremos el método de descenso de subgradiente usando un tamaño de paso $\alpha_i = \alpha$ constante para todo i. Sea x^* una solución óptima de* (A.1) *y supongamos que los gradientes son uniformemente acotados,es decir, $\left\|g^i\right\|_2 \leq G$ para todo $i \geq 0$. Entonces para todo $k \geq 0$, se cumple la siguiente desigualdad:*

$$\min_{i \in \{0,\ldots,k\}} f\left(x^i\right) \leq f^* + \frac{\left\|x^0 - x^*\right\|_2^2}{2(k+1)\alpha} + \frac{\alpha G^2}{2}. \tag{A.3}$$

Un método similar a los anteriores para funciones dos veces diferenciables es el método de Newton. Consideremos la aproximación de segundo orden $\hat{f}$ de la función convexa f en el punto x

$$\hat{f}(x+v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v.$$

Esta es una función cuadrática convexa de v que se minimiza en $v = -\nabla^2 f(x)^{-1} \nabla f(x)$. Se tiene entonces un método muy similar al descenso de gradiente pero, en este caso, la actualización viene dada por

$$x^{k+1} \leftarrow x^k - \nabla^2 f(x)^{-1} \nabla f(x).$$

Bibliografía

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[FGM17] Robert M. Freund, Paul Grigas, and Rahul Mazumder. A new perspective on boosting in linear regression via subgradient optimization and relatives. *Ann. Statist.*, 45(6):2328–2364, 2017.

[FHT00] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Ann. Statist.*, 28(2):337–407, 2000. With discussion and a rejoinder by the authors.

[MBBF00] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 512–518. MIT Press, 2000.