Gradient Boosting

Alondra Elizabeth Matos Mendoza, María Guadalupe López Salomón
Centro de Investigación en Matemáticas. Unidad Monterrey
Email: alondra.matos@cimat.mx, maria.salomon@cimat.mx

***Resumen*—En este reporte se detalla el algoritmo de Gradient Boosting, así como una versión mejorada y regularizada llamada Extreme Gradient Boosting (XGBoost), que ofrece mayor eficiencia y rendimiento. Además,se presenta una implementación de estos métodos en un caso de estudio sobre la clasificación de pacientes con falla cardíaca.**

I. Introducción

En aplicaciones de minería de datos industriales y comerciales, existen desafíos únicos que los procedimientos de aprendizaje deben abordar. Estos desafíos surgen de las características de los conjuntos de datos, que suelen ser grandes y complejos, con una variedad de variables y observaciones. Los datos pueden estar desordenados, con una combinación de variables cuantitativas, binarias y categóricas, y la presencia de valores faltantes. Las distribuciones de las variables pueden ser asimétricas y con colas largas, y es común encontrar errores de medición y valores atípicos. Además, las variables predictoras pueden tener escalas muy diferentes. En resumen, los desafíos en estas aplicaciones se relacionan con el tamaño y complejidad de los datos, las limitaciones computacionales, la diversidad de variables y la presencia de valores faltantes y atípicos.

Los árboles de decisión son rápidos, interpretables y adaptables en el manejo de diferentes tipos de variables y valores faltantes. También son robustos frente a transformaciones y valores atípicos, y realizan selección de características. Sin embargo, su precisión puede no ser óptima, aunque se mejora significativamente mediante el uso de Boosting Trees. Esto implica sacrificar algunas ventajas como velocidad, interpretabilidad y robustez en ciertos escenarios, como distribuciones de clases superpuestas y errores en la etiquetación de los datos de entrenamiento (especialmente en el caso de AdaBoost). Para abordar estos problemas, se desarrollaron los modelos basados en Gradient Boosting.

II. Boosting Trees

En los árboles de regresión y clasificación, el espacio de todos los valores de las variables predictoras se divide en j regiones disjuntas representadas por los nodos terminales del árbol, denotadas como R_j. A cada región se le asigna una constante γ_j. Definiendo $f(x)$ como una función que devuelve los valores predichos, la regla predictiva se define de la siguiente manera:

$$x \in R_j \rightarrow f(x) = \gamma_j$$

Por lo tanto, un árbol es expresado formalmente como:

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I\left(x \in R_j\right)$$

con los parámetros $\theta = \{R_j, \gamma_j\}_1^J$, en donde J es tratado comúnmente como un meta-parámetro.

Los parámetros se determinan minimizando el riesgo empírico,

$$\hat{\Theta} = arg \min_{\Theta} \sum_{j=1}^{J} \sum_{x \in R_j} L(y_i, \gamma_j)$$

donde y_i son los valores observados de la muestra y L es la función de costo.

Inducido de manera progresiva hacia adelante, el modelo de Boosted Tree es la suma de tales árboles,

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m), \tag{1}$$

de tal manera que la predicción final $f_M(x)$ es la suma de las predicciones de cada árbol y en cada paso, se debe resolver:

$$\hat{\Theta}_m = arg \min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m)\right)$$

para el conjunto de regiones y constantes $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ del siguiente árbol, dado el modelo previo $f_{m-1}(x)$.

Dadas las regiones R_{jm}, encontrar las constantes óptimas γ_{jm} en cada región suele ser típicamente sencillo:

$$\hat{\gamma}_{jm} = arg \min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_m - 1(x_i) + \gamma_{jm}\right)$$

En Adaboost, los errores de los modelos débiles existentes se identifican asignando pesos más altos a los puntos de datos. Estos puntos de datos con pesos altos se consideran desafiantes o difíciles de predecir con precisión. Al enfocarse en estos puntos, Adaboost busca mejorar el rendimiento general del ensemble al prestar más atención a las observaciones con las que los modelos débiles anteriores tuvieron dificultades. Los modelos débiles posteriores se diseñan para abordar

estas deficiencias específicas y mejorar la precisión predictiva global del ensemble.

En Gradient Boosting, las deficiencias se identifican mediante el análisis de los gradientes. Los gradientes representan la dirección y magnitud del error o residual para cada observación en el conjunto de datos. Al examinar estos gradientes, el algoritmo puede determinar qué áreas del modelo deben mejorarse o ajustarse para minimizar el error general y optimizar las predicciones. Los gradientes guían el proceso iterativo de construcción de modelos débiles subsiguientes. Al minimizar continuamente los errores, el Gradient Boosting crea gradualmente un modelo de conjunto sólido que puede hacer predicciones precisas.

III. Gradient Boosting

El Gradient Boost empieza con un único nodo terminal, en vez de un árbol como en Adaboost. En particular, comienza inicializando el modelo con un valor constante γ, que representa la predicción inicial para todos los elementos de la muestra. Dicho valor es aquél que minimiza el costo total, que es la suma de los valores de la función de costo de los elementos de la muestra. Posteriormente, se determina un número total de árboles a construir M (generalmente $M = 100$) y para cada árbol, se procede a realizar lo siguiente:

1. Se calculan los *Pseudo residuales* r_{im}, los cuales se obtienen multiplicando el gradiente por -1. El nombre se debe a que la derivada de la función de costo con respecto al valor predicho resulta en un valor similar a un residuo. Por ejemplo. si se considerara la función de costo $L(y_i, f(x_i)) = (y_i - f(x_i))^2$, se tendría que $\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} = -2(y_i - f(x_i))$.

2. Se ajusta un árbol de regresión a los pseudo residuales y se crean las regiones terminales R_{jm}, $(j = 1, ..., J_m)$.

3. Se determinan los valores de salida para cada nodo terminal. El valor de salida es el valor γ_{jm} que minimiza la suma de los valores de la función de costo para los pseudo residuales de los elementos de la muestra que se encuentran en el nodo j, considerando la predicción previa.

4. Para cada elemento de la muestra, se crea una nueva predicción, el cual es la suma de la predicción anterior más el producto de la suma de los valores de salida de cada nodo en el que se haya terminado por la tasa de aprendizaje ν $(0 \leq \nu \leq 1)$, que reduce el efecto que cada árbol tiene al final de la predicción, mejorando la exactitud a largo plazo.

El Algoritmo 1 muestra el esquema general para este método.

Algorithm 1 Gradient Tree Boosting

Entrada Datos $\{(x_i, y_i)\}_{i=1}^n$ y una función de costo $L(y_i, f(x))$diferenciable.
Inicializar $f_0(x) = argmin_\gamma \sum_{i=1}^N L(y_i, \gamma)$
for $m = 1 : M$ **do**
  for $i = 1, 2, ..., N$ **do**
    Calcular $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$
  end for
  Ajustar un árbol de regresión a los valores r_{im} y crear las regiones terminales R_{jm} $j = 1, 2, ..., J_m$
  for $j = 1, 2, ..., J_m$ **do**
    Calcular $\gamma_{jm} = arg \ \min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right)$
  end for
  Actualizar $f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I\left(x \in R_{jm}\right)$
end for
Salida $\hat{f}(x) = f_M(x)$

Cabe mencionar que, para regresión, la función de costo más utilizada es la del error cuadrático $L(y_i, f(x_i)) = \frac{1}{2}\left(y_i - f(x_i)\right)^2$, en donde $\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} = -(y_i - f(x_i)$. Con esta función, la predicción inicial es el promedio de los valores de la variable dependiente, el valor r_{im} es igual al residuo $y_i - f(x_i)$ y $\hat{\gamma}_{jm}$ es el promedio de esos residuales correspondientes a la región R_{jm}.

III-A. Gradient Tree Boosting para clasificación

Considerando que, cuanto mejor sea la predicción, mayor es la verosimilitud, se busca maximizar la función de verosimilitud de la muestra observada, lo cual es equivalente a minimizar el negativo del logaritmo de la función de verosimilitud. En consecuencia, la función de costo que se utiliza cuando se usa el algoritmo de Gradient Boosting para clasificación binaria es el negativo de la log-verosimilitud de la binomial (también conocida como desviación o entropía cruzada):

$$\begin{aligned} L(y, p(x)) &= -\sum_{i=1}^{N} y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i)) \\ &= \sum_{i=1}^{N} -y_i \left[\log p(x_i) - \log(1 - p(x_i))\right] - \log(1 - p(x_i)) \\ &= \sum_{i=1}^{N} -y_i \log \frac{p(x_i)}{(1 - p(x_i))} - \log(1 - p(x_i)) \end{aligned}$$

en donde $p(x_i)$ es la probabilidad predicha para x_i.

Además, denotando la transformada logit como $\log(odds) = \log \frac{p(x_i)}{(1-p(x_i))}$, se obtiene que $p(x_i) = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$, por lo que a partir del logaritmo de la razón de probabilidades se pueden obtener las probabilidades mediante una transformación con la función logística. Entonces, $\log(1 - p(x_i)) = \log(1 - \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}) = -\log(1 + e^{\log(odds)})$. Por lo tanto, interpretando a f como la transformada logit, la función de costo se puede reescribir como:

$$L(y, f(x)) = \sum_{i=1}^{N} -y_i f(x) + \log(1 + e^{f(x)})$$

Entonces, los pseudo residuales se obtienen a partir de:

$$\begin{aligned} -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} &= y_i - \frac{e^{f(x_i)}}{1 + e^{f(x_i)}} \\ &= y_i - p(x_i) \end{aligned}$$

Cabe mencionar que la predicción inicial $f_0(x)$ para cada elemento de la muestra es el *log(odds)* basado en los valores de la muestra.

Para obtener de manera rápida y sencilla las constantes óptimas γ_{jm}, que se calculan luego de ajustar el árbol de regresión a los residuales y de crear las regiones $R_i m$, se utiliza la aproximación de la función de costo mediante el polinomio de Taylor de segundo orden, de tal forma que:

$$\begin{aligned} L\left(y_i, f_{m-1}(x_i) + \gamma\right) \approx\ & L\left(y_i, f_{m-1}(x_i)\right) \\ & + \frac{\partial}{\partial f()} L\left(y_i, f_{m-1}(x_i)\right) \gamma \\ & + \frac{1}{2} \frac{\partial^2}{\partial f()^2} L\left(y_i, f_{m-1}(x_i)\right) \gamma^2 \end{aligned}$$

Por lo tanto, para cada región j se tiene que:

$$\begin{aligned} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right) \approx & \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i)\right) \\ & + \left[\sum_{x_i \in R_{jm}} \frac{\partial}{\partial f()} L\left(y_i, f_{m-1}(x_i)\right)\right] \gamma \\ & + \frac{1}{2}\left[\sum_{x_i \in R_{jm}} \frac{\partial^2}{\partial f()^2} L\left(y_i, f_{m-1}(x_i)\right)\right] \gamma^2 \end{aligned} \quad (2)$$

Derivando con respecto a γ, se sigue que:

$$\begin{aligned} \frac{\partial}{\partial \gamma} L\left(y_i, f_{m-1}(x_i) + \gamma\right) \approx & \sum_{x_i \in R_{jm}} \frac{\partial}{\partial f()} L\left(y_i, f_{m-1}(x_i)\right) \\ & + \left[\sum_{x_i \in R_{jm}} \frac{\partial^2}{\partial f()^2} L\left(y_i, f_{m-1}(x_i)\right)\right] \gamma \end{aligned}$$

Igualando la derivada a 0 y resolviendo para γ, se obtiene que la constante óptima para la j-ésima región es:

$$\begin{aligned} \hat{\gamma}_{jm} &= \frac{-\left[\sum_{x_i \in R_{jm}} \frac{\partial}{\partial f()} L\left(y_i, f_{m-1}(x_i)\right)\right]}{\sum_{x_i \in R_{jm}} \frac{\partial^2}{\partial f()^2} L\left(y_i, f_{m-1}(x_i)\right)} \\ &= \frac{\sum_{x_i \in R_{jm}} y_i - p(x_i)}{\sum_{x_i \in R_{jm}} p(x_i)(1 - p(x_i))} \end{aligned}$$

El problema de clasificación binaria se puede extender a clasificación multiclase donde y_{ik} es igual a 1 si la observación i está en la clase k y 0 de otra forma, utilizando la función de costo de desviación multinomial y la transformación simétrica con la función logística. En este caso, como se muestra en el Algoritmo 2, se construyen K árboles (uno para cada clase) en cada iteración.

Algorithm 2 Gradient Tree Boosting para clasificar K clases

1: Inicializar $f_{k0}(x) = 0,\ k = 1, 2, ..., K$
2: **for** $m = 1$ *to* M **do**
3: Establecer $p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}} \quad k = 1, 2, ..., K$
4: **for** $k = 1$ *to* K **do**
5: Calcular $r_{im} = y_{ik} - p_k(x_i) \quad i = 1, 2, ...N$
6: Ajustar un árbol de regresión a los valores r_{ikm}, $i = 1, 2, ..., N$, y crear las regiones terminales R_{jkm} $j = 1, 2, ..., J_m$
7: Calcular

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} r_{ikm}}{\sum_{x_i \in R_{jkm}} |r_{ikm}|(1 - |r_{ikm}|)}, \ \ j = 1, 2, ..., J_m$$

8: Actualizar $f_{km}(x) = f_{k,m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jkm} I\left(x \in R_{jkm}\right)$
9: **end for**
10: **end for**
11: **Salida** $\hat{f}_k(x) = f_{kM}(x), \quad k = 1, 2, ..., K$

IV. Extreme Gradient Boosting

XGBoost o *Extreme Gradient Boost* es una forma regularizada de *gradient boosting* con muchas ventajas diferentes, que surge a partir de la necesidad evidente de algoritmos más rápidos para el manejo del big data. Dentro de los avances clave, se distinguen la ganancia de rapidez, el manejo de datos, así como mejoras significativas en la precisión.

La estructura general descrita en el Algoritmo 1, al igual que los hiperparámetros del *gradient boosting* tradicional se incorporan en *XGBoost*.

Considerando un modelo de Boosted Tree (Eq. 1) con M árboles para un conjunto de datos dado con N ejemplos y K características $D = \{(\boldsymbol{x}_i, y_i\} \ \left(|\mathcal{D}| = N, \boldsymbol{x}_i \in \mathbb{R}^K, y_i \in \mathbb{R}\right)$ y sean:

- $T(x; \Theta_m)$: m-ésimo árbol de regresión con parámetros $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^J$
- $|T_m|$: Número de nodos terminales (hojas) en el m-ésimo árbol.
- γ_m: Valores de salida (pesos/scores) asociados a la hojas del m-ésimo árbol.

El objetivo del XGBoost es encontrar los valores de salida γ de las hojas que minimicen la siguiente función objetivo regularizada:

$$\mathbf{L}(f_M) = \left[\sum_{i=1}^{n} L(y_i, f(x))\right] + \sum_{m=1}^{M} \Omega(T(x; \Theta_m)) \quad (3)$$

$$donde\ \Omega(T(x; \Theta_m)) = \alpha |T_m| + \frac{1}{2} \lambda ||\gamma_m||^2$$

El término Ω penaliza la complejidad del modelo, es decir, es decir, la estructura de los árboles de regresión. En particular, el parámetro de penalización α fomenta la poda. El término adicional de regularización ayuda a suavizar los pesos finales aprendidos para evitar el sobreajuste. Si $\lambda > 0$, los valores de salida ω se encogen acercándose a 0.

Cuando el parámetro de regularización se establece en cero, el objetivo vuelve al enfoque de Gradient Boost convencional.

Como el modelo es entrenado de manera aditiva, la minimización se realiza por iteración, ; de tal forma que, sea $f_m(x_i)$ la predicción de la í-esima instancia en la m-ésima iteración, el vector de salida γ_{jm} de la j-ésima hoja del m-ésimo árbol se encuentran minimizando la siguiente función objetivo:

$$\mathbf{L}^{(m)} = \sum_{j=1}^{J_m} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) + \alpha|T_m| + \frac{1}{2}\lambda \sum_{j=1}^{J_m} \gamma_{jm}^2 \tag{4}$$

Tanto para clasificación como regresión, la predicción inicial puede ser cualquier valor. Sin embargo, por default, es igual a 0,5.

En comparación al Gradient Boost, XGBoost usa la aproximación mediante el polinomio de Taylor de segundo grado (Eq. 2) para resolver rápidamente el problema de optimización (Eq. 4), tanto para regresión como clasificación.

$$\begin{aligned}\mathbf{L}^{(m)} \approx \sum_{j=1}^{J_m} \sum_{x_i \in R_{jm}} & L\left(y_i, f_{m-1}(x_i)\right) \\ & + \sum_{x_i \in R_{jm}} \frac{\partial}{\partial f(x_i)} L\left(y_i, f_{m-1}(x_i)\right) \gamma_{jm} \\ + \sum_{x_i \in R_{jm}} \frac{1}{2} \frac{\partial^2}{\partial f(x_i)^2} & L\left(y_i, f_{m-1}(x_i)\right) \gamma_{jm}^2 + \alpha|T_m| + \frac{1}{2}\lambda \sum_{j=1}^{J_m} \gamma_{jm}^2\end{aligned}$$

Si se denota el gradiente como g y la matriz hessiana como h, y se eliminan los términos constantes que no tienen efecto en los valores de salida, la función objetivo se simplifica a:

$$\begin{aligned}\bar{\mathbf{L}}^{(m)} &= \sum_{j=1}^{J_m} \sum_{x_i \in R_{jm}} \left[g_i \gamma_{jm} + \frac{1}{2} h_i \gamma_{jm}^2 \right] + \frac{1}{2}\lambda \sum_{j=1}^{J_m} \gamma_{jm}^2 \\ &= \sum_{j=1}^{J_m} \left[\left(\sum_{x_i \in R_{jm}} g_i \right) \gamma_{jm} + \frac{1}{2} \left(\sum_{x_i \in R_{jm}} h_i + \lambda \right) \gamma_{jm}^2 \right] \\ &\quad + \alpha|T_m|\end{aligned} \tag{5}$$

Derivando la expresión anterior con respecto a γ_{jm}, se obtiene que:

$$\frac{\partial}{\partial \gamma_{jm}} \bar{\mathbf{L}}^{(m)} = \sum_{x_i \in R_{jm}} g_i + \left(\sum_{x_i \in R_{jm}} h_i + \lambda \right) \gamma_{jm} \tag{6}$$

Igualando a 0 y resolviendo para γ_{jm}, se obtiene que la constante óptima para la j-ésima región es:

$$\gamma_{jm} = -\frac{\sum_{x_i \in R_{jm}} g_i}{\sum_{x_i \in R_{jm}} h_i + \lambda}$$

Entonces, de la Eq. 5, se obtiene que el valor óptimo del m-ésimo árbol completo es:

$$\bar{\mathbf{L}}^{(m)}_{similaridad} = -\frac{1}{2} \sum_{j=1}^{J_m} \frac{\left(\sum_{x_i \in R_{jm}} g_i \right)^2}{\sum_{x_i \in R_{jm}} h_i + \lambda} + \alpha|T| \tag{7}$$

La Eq. 7 puede ser utilizada como una función de puntuación para medir la calidad de la estructura del m-ésimo árbol (*score de similaridad*). Este puntaje es comparable al puntaje de impureza utilizado para evaluar los árboles de decisión, aunque se aplica a una gama más amplia de funciones objetivo.

Por lo general, resulta inviable enumerar todas las estructuras posibles de un árbol. En cambio, se emplea un algoritmo voraz que parte de una sola hoja y añade ramas al árbol de manera iterativa. Considerando I_L e I_R como los conjuntos de instancias correspondientes a los nodos izquierdo y derecho después de la partición, entonces $I = I_L \cup I_R$ y la ganancia (reducción de costo) después del split viene dado por:

$$\begin{aligned}Gain_{split} &= left_{similarity} + right_{similarity} - root_{similarity} \\ &= \frac{1}{2} \left[\frac{\left(\sum_{x_i \in I_L} g_i \right)^2}{\sum_{x_i \in I_L} h_i + \lambda} + \frac{\left(\sum_{x_i \in I_R} g_i \right)^2}{\sum_{x_i \in I_R} h_i + \lambda} \right] \\ &\quad - \frac{1}{2} \frac{\left(\sum_{x_i \in I} g_i \right)^2}{\sum_{x_i \in I} h_i + \lambda} - \alpha\end{aligned} \tag{8}$$

La Eq. 8 se emplea para evaluar posibles candidatos de splits (divisiones).

En particular, el Cuadro I muestra los valores óptimos para regresión y clasificación binaria cuando se considera como función de costo el error cuadrático y el negativo de la log-verosimilitud, respectivamente.

	Regresión	Clasificación binaria
Función de costo	$\frac{1}{2}(y_i - f(x_i))^2$	$-y_i \log p_i$ $-(1-y_i)\log(1-p_i)$
Valores de salida (γ_{jm})	$\frac{\sum_{x_i \in R_{jm}} y_i - f(x_i)}{\sum_{x_i \in R_{jm}} 1+\lambda}$	$\frac{\sum_{x_i \in R_{jm}} y_i - p_i}{\sum_{x_i \in R_{jm}} p_i(1-p_i)+\lambda}$
Score de similaridad	$\frac{1}{2}\frac{\sum_{x_i \in R_{jm}} (y_i - f(x_i))^2}{\sum_{x_i \in R_{jm}} 1+\lambda}$ + $\alpha\|T\|$	$\frac{1}{2}\frac{\sum_{x_i \in R_{jm}} (y_i - p_i)^2}{\sum_{x_i \in R_{jm}} p_i(1-p_i)+\lambda}$ + $\alpha\|T\|$

Cuadro I
VALORES ÓPTIMOS PARA REGRESIÓN Y CLASIFICACIÓN BINARIA

Submuestreo de columna y contracción

Para evitar aún más el sobreajuste, se implementan dos métodos adicionales. El primer método se denomina contracción. La contracción vuelve a escalar los pesos de los componentes recién agregados por un factor ν después de cada iteración. Esta técnica es similar al uso de una tasa de aprendizaje en la optimización estocástica, ya que disminuye el impacto de los árboles individuales, dejando espacio para que árboles futuros mejoren el modelo.

La segunda técnica es el submuestreo de columnas (características). Según los comentarios de los usuarios, la incorporación del submuestreo de columnas como medida de prevención contra el sobreajuste demuestra ser más eficaz que la técnica tradicional de submuestreo de filas (que también está disponible). Además, el uso de submuestras de columna conduce a cálculos más rápidos en el algoritmo paralelo que se describe más adelante.

IV-A. Algoritmos de búsqueda de splits

Uno de los principales desafíos en el aprendizaje de árboles es determinar la división óptima, como se describe en Eq.8. Para lograr esto, un algoritmo de búsqueda de división examina todas las posibles divisiones en todas las características. Este algoritmo se conoce como el algoritmo voraz exacto, el cual se describe en el Algoritmo 3. Para lograr un rendimiento eficiente, el algoritmo debe ordenar los datos en función de los valores de la variable y luego procesar los datos en orden para acumular estadísticas de gradiente para la puntuación de la estructura del árbol en la Eq. 8.

El algoritmo voraz exacto es muy eficaz, ya que explora sistemáticamente todos los posibles puntos de división. Sin embargo, se vuelve poco práctico cuando los datos no caben completamente en la memoria, lo que hace imposible una enumeración eficiente. Por lo tanto, es necesario un algoritmo aproximado.

La descripción del método aporximado de búsqueda de splits se muestra en el Algoritmo 4. Dicha técnica comienza sugiriendo posibles puntos de división basados en los percentiles de la distribución de la variable (característica). Estos puntos candidatos se utilizan para dividir las características continuas en grupos. Luego, el algoritmo combina las estadísticas dentro de cada grupo y selecciona la solución óptima de los splits propuestos con base en las estadísticas agregadas.

Hay dos variantes del algoritmo aproximado según el conjunto de splits propuesto: el global y el local. La variante global propone todas los splits candidatos al principio y las utiliza en todos los niveles; mientras que, la variante local propone después de cada división. El método global requiere menos pasos de propuesta pero más puntos candidatos. El método local refina los candidatos después de las divisiones y es mejor para árboles más profundos. No obstante, con suficientes candidatos, la variante global puede ser igualmente precisa.

Algorithm 3 Algoritmo voraz exacto para la búsqueda de splits

Entrada: I, conjunto de instancias del nodo actual,
Entrada: d, número de variables (características/columnas)
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1 \ : \ K$ **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for $j \in (I, ordenado\ por\ x_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max\left(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right)$
  end for
end for
Salida: Split con el máximo score de ganancia

Algorithm 4 Algoritmo aproximado para la búsqueda de splits

for $k = 1 \ : \ K$ **do**
  Proponer $S_k = \{s_{k1}, s_{k2}, ..., s_{kl}\}$, donde S_k es el conjunto de percentiles de la variable k
  El conjunto S_k puede proponerse por árbol (global) o por split (local)
end for
for $k = 1 \ : \ K$ **do**
  $G_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} g_i$
  $H_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} h_i$
end for
Seguir el mismo paso que en la sección anterior para encontrar max puntuar solo entre los splits propuestas

IV-A1. Weighted Quantile Sketch: Cuando la base de datos es muy grande, ordenar una lista de números y encontrar los cuantiles se vuelve tardado. Sin embargo, un algoritmo basado en sketch puede rápidamente aproximar soluciones. En particular, el XGBoost utiliza el Weighted Quantile Sketch para obtener una representación aproximada de la distribución de la variable.

Dicho algoritmo divide los datos en subconjuntos más pequeños mediante *programación paralela* y combina los valores para obtener un histograma aproximado, el cual es utilizado para obtener los cuantiles aproximados.En particular, cada observación tiene un peso asociado, de tal forma que la suma de los pesos son iguales en cada cuantil. Específicamente, dicho peso se obtiene evaluando la segunda derivada de la función de costo.

La ventaja de utilizar los cuantiles ponderados es que crean quantiles más pequeños cuando es necesario, pues dividen el histograma colocando las observaciones con predicciones de baja confianza en cuantiles con menos observaciones.

Para regresión, considerando la función de costo del error cuadrático, los pesos son iguales a 1, por los que los cuantiles contienen el mismo número de observaciones. Por el contrario, contemplandola desviación como función de costo, los pesos en clasificación binaria son el producto de la probabilidad previa de éxito por la probabilidad previa de fracaso.

IV-B. Sparsity-aware Split Finding

El algoritmo Sparsity-aware Split Finding se utiliza para construir árboles con datos faltantes y cómo tratar con nuevas observaciones cuando hay valores no disponibles.

Particularmente, cuando falta un valor, la instancia se clasifica en la dirección predeterminada. En cada rama hay dos opciones de dirección predeterminada. Las direcciones predeterminadas óptimas se aprenden a partir de los datos.

La mejora principal del Sparsity-aware Split Finding, el cual se describe en el Algoritmo 5, consiste en centrarse únicamente en las entradas no faltantes I_k. El algoritmo descrito considera la ausencia de un valor como un valor faltante y determina el enfoque óptimo para manejar los valores faltantes. Este mismo algoritmo también se puede aplicar cuando la ausencia corresponde a un valor especificado por el usuario, con la limitación de que solo se consideran soluciones consistentes durante la enumeración.

Algorithm 5 Sparsity-aware Split Finding

Entrada: I, conjunto de instancias del nodo actual
Entrada: $I_k = \{i \in I | x_{ik} \neq missing\}$
Entrada: d, número de variables (características/columnas)
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1 \ : \ K$ **do**
  Los valores faltantes se clasifican a la derecha.
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for $j \in (I_k, ordenado\ ascendentemente\ por\ x_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max\left(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right)$
  end for
  Los valores faltantes se clasifican a la izquierda.
  $G_R \leftarrow 0$, $H_R \leftarrow 0$
  for $j \in (I_k, ordenado\ descendentemente\ por\ x_{jk})$ **do**
    $G_R \leftarrow G_R + g_j$, $H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R$, $H_L \leftarrow H - H_R$
    $score \leftarrow \max\left(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right)$
  end for
end for
Salida: Split y dirección predeterminada con el máximo score de ganancia

IV-C. Diseño computacional

IV-C1. Acceso con reconocimiento de caché: La CPU (unidad central de procesamiento) se encuentra conectada a las siguientes unidades de memoria:

- Memoria caché: La cpu puede usar esta memoria más rápido que cualquier otra memoria en la computadora.
- Memoria principal: Mientras esta memoria es más grande que la memoria caché, se tarda más en usar.
- Disco duro: Puede almacenar la mayoría de cosas, pero es la memoria más lenta.

Para lograr un alto rendimiento en un programa, se busca maximizar la utilización de la memoria caché en la mayoría de los procesos. Así que, el XGBoost aprovecha la memoria caché almacenando los gradientes y las matrices hessianas para calcular de manera rápida los scores de similitud y los valores de salida.

IV-C2. Bloques para computación fuera del núcleo: Cuando la base de datos es demasiado grande para caber en la memoria caché y la memoria principal, se hace necesario almacenar una parte de ella en el disco duro. Sin embargo, como la lectura y escritura de datos en el disco duro es lenta, XGBoost busca minimizar estas operaciones comprimiendo los datos. Incluso cuando la CPU requiere tiempo para descomprimir los datos, el proceso sigue siendo más rápido en comparación con el tiempo que tarda el disco duro en leer los datos.

Además, cuando se dispone de más de un disco duro, XGBoost utiliza la técnica de *Sharding* para acelerar el acceso a los datos en disco. El sharding implica distribuir los datos en múltiples discos duros de manera equilibrada, lo que permite realizar operaciones de lectura y escritura de manera paralela en varios discos simultáneamente.

V. Aplicación.

Se ajustaron modelos para predecir si un paciente sufrirá una falla cardíaca, con el fin de realizar una comparación entre el Gradient Boosting y el XGBoost, junto con otros métodos de clasificación como árboles de decisión, Adaboost, random forests y redes neuronales.

La implementación de los algoritmos *Gradient Boosting* y *XGBoost*, respectivamente, se llevaron a cabo en un entorno basado en macOS Monterrey Versión 12.3 con un equipo cuyo procesador corresponde al Apple M1 Pro y 16 Gb de memoria Ram. Se utilizó Google Colab como el entorno de desarrollo en línea, con Python 3 como el lenguaje de programación principal. Estas configuraciones proporcionaron un ambiente adecuado y eficiente para llevar a cabo la implementación de los algoritmos.

La base de datos utilizada, denominada *Heart failure clinical records*, se obtuvo de https://archive.ics.uci.edu/dataset/519/heart+failure+clinical+records. Dicha base se compone de registros médicos de 299 pacientes que sufrieron falla cardíaca, recolectados durante su seguimiento médico, en donde el perfil de cada paciente tiene 13 características clínicas, las cuales se describen a continuación.

- 'Edad': Edad del paciente en años.
- 'Anemia': Decremento de las células rojas o hemoglobina (booleana).
- 'Creatinina fosfocinasa(CPK)': Nivel de la enzima (CPK) en la sangre (mcg/L).
- 'Diabetes': Si el paciente tiene diabetes (booleana).

- ‘Fracción de eyección’: Porcentaje de sangre que sale del corazón en cada contracción (porcentaje).
- ‘Hipertensión’: Si el paciente padece de hipertensión.
- ‘Plaquetas’: Plaquetas en la sangre (kiloplaquetas/mL).
- ‘Sexo’: Mujer u hombre (binaria).
- ‘Suero de creatinina’: Nivel de sodio sérico en la sangre (mEq/L).
- ‘Fumador’: Si el paciente es fumador o no (booleana).
- ‘Tiempo’: Periodo de seguimiento (días).
- ‘Falla cardiaca’: Si el paciente falleció durante el periodo de seguimiento (booleana)[Variable objetivo].

Se dividió el conjunto de datos en un $70\,\%$ para el ajuste de los modelos y un $30\,\%$ para realizar las pruebas y la evaluación de los modelos. En particular, el conjunto de entrenamiento estaba compuesto por un $71{,}77\,\%$ de registros de personas sin falla cardíaca y un $28{,}22\,\%$ de registros de personas que sí la sufrieron, por lo que había un desequilibrio en la distribución de las clases, ya que una de ellas estaba sobrerepresentada en comparación con la otra.

Las librerías necesarias para la implementación de cada uno de los modelos de clasificación son: DecisionTreeClassifier, AdaBoostClassifier, GradientBoostingClassifier, XBGClassifier, RandomForestClassifier y MLPClassifier.

Es importante destacar que se empleó la técnica de GridSearch para encontrar los mejores hiperparámetros de cada modelo, realizando una búsqueda exhaustiva de todas las combinaciones posibles de hiperparámetros dentro de una cuadrícula predefinida, lo que permite encontrar la configuración óptima de hiperpárametros que maximiza el desempeño de cada algoritmo. Además, se llevó a cabo la validación cruzada para evaluar los algoritmos en múltiples subconjuntos de prueba, lo cual permitió obtener una puntuación más confiable y robusta. Se estableció que el número de folds sea igual a 5.

Cabe mencionar que la métrica de evaluación utilizada fue el F1 score ponderado. Específicamente, se empleó dicha métrica sobre la F1 tradicional, dado que esta toma en cuenta el desequilibrio de las clases al calcular el promedio ponderado, es decir, que asigna un peso proporcional al tamaño de cada clase, lo que significa que las clases más pequeñas tienen una influencia mayor en el resultado final que las clases más grandes. Lo que ayuda a evitar que se sesgue hacia la clase mayoritaria, que en este caso correspondía a la clase 0.

En el cuadro **??** se presentan los valores óptimos encontrados para los hiperparámetros.

Modelo	Parámetro asociado	Descripción
Decision Tree Classifier	max_depth = None	Especifica la profundidad máxima del árbol de decisiones. Controla el número máximo de divisiones o niveles que el árbol puede tener. Un valor bajo, como 2, indica un árbol poco profundo con decisiones simples.
AdaBoost Classifier	learning rate = 0.05	Controla la distribución de cada clasificador débil en el conjunto. Un learning rate más bajo disminuye la influencia de cada clasificador débil.
	n_estimators = 300	Especifica el número de clasificadores débiles utilizados en el conjunto. Aumentar el número de estimadores puede mejorar el rendimiento del modelo, pero hay un punto de rendimiento óptimo que depende de los datos.
GradientBoosting Classifier	learning rate = 0.4	Controla la contribución de cada árbol en el conjunto. Un learning rate más alto aumenta la contribución cada árbol, lo que puede llevar a un ajuste mayor de los datos de entrenamiento.
	n_estimators = 300	Especifica el número de árboles de decisión utilizados en el conjunto.
	max_depth = 8	Profundidad máxima de los estimadores de regresión individuales.
	max_features = 2	El número de características a considerar al buscar la mejor división.
XGBoost	learning rate = 0.5	Al igual que en GradientBoosting, controla la contribución de cada árbol en el conjunto.
	n_estimators = 400	Especifica el número de árboles de decisión utilizados en el conjunto.
	max_depth = 5	La profundidad máxima de un árbol.
	min_child_weight = 8	Define la suma mínima de pesos de todas las observaciones requeridas en un niño,
	gamma = 1	Especifica la reducción de pérdida mínima necesaria para realizar una división.
RandomForest	max_depth = 6	Especifica la profundidad máxima de los árboles de decisión en el bosque aleatorio.
	n_estimators = 50	Especifica el número de árboles de decisión en el bosque aleatorio.
MLPClassifier	alpha = 5	Es el parámetro de regularización L2 que controla la penalizaciónpor la complejidad del modelo. Un valor más alto de alpha penalizará más los pesos grandes y ayudará a prevenir el sobreajuste.
	hidden_layer_sizes = 50	Especifica el número de neuronas en cada capa oculta de la red neuronal MLP.
	max_iter = 5000	Especifica el número máximo de iteraciones (pasadas por los datos de entrenamiento) para entrenar el modelo.
	solver = sgd	Es el algoritmo utilizado para optimizar los pesos durante el entrenamiento dela red. ‘sed’ representa descenso de gradiente descendente estocástico.

Cuadro II
Descripción de los hiperparámetros

En la Figura 1 se representa el F1-Score y el tiempo de ejecución obtenidos por los diferentes modelos. Se puede observar que los algoritmos de Gradient Boosting y XGBoost presentan un mejor desempeño en este conjunto de datos en comparación con los otros modelos implementados. Específicamente, XGBoost muestra el mejor rendimiento en términos de la relación calidad-tiempo de ejecución.

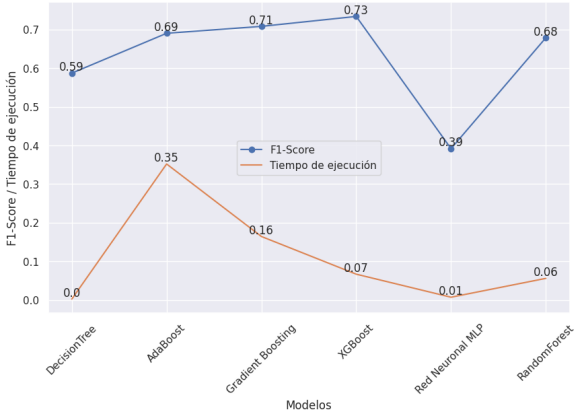

Figura 1. Rendimiento y tiempo de ejecución de GBoosting y XGBoost vs otros métodos de clasificación

Cabe destacar que el modelo basado en Gradient Boosting muestra un mejor desempeño cuando es regularizado. La Figura 2 ilustra el efecto de diferentes estrategias de regularización para Gradient Boost. La implementación de la

regularización mediante la técnica de contracción, que ajusta la contribución de cada árbol utilizando una tasa de aprendizaje de 0.4, en combinación con el submuestreo aleatorio de características (*max_features=2*), produce una mejora notable en el rendimiento del modelo. Además, se observa que aplicar un submuestreo de las instancias, puede generar modelos más precisos al reducir la varianza. El proceso de realizar el muestreo sin reemplazo de una fracción de las observaciones de entrenamiento y utilizar esa submuestra para construir el siguiente árbol se conoce como *stochastic gradient boosting*.

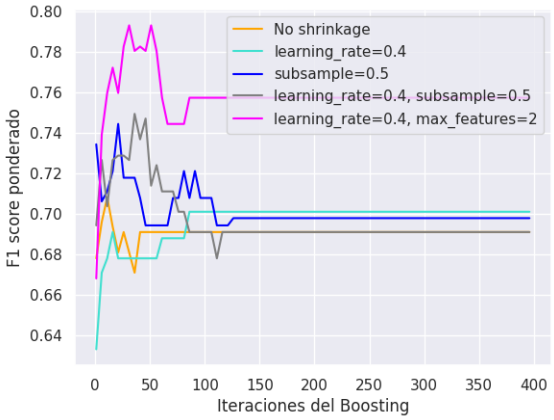

Figura 2. Efecto de regularización en Gradient Boosting

En la Figura 3 se presenta la importancia de las variables predictoras para el modelo de XGBoost. Se puede observar que el tiempo es el más relevante, seguido por el suero de creatina. Por otro lado, la diabetes, la hipertensión y el hecho de fumar no parecen tener influencia en la ocurrencia de una falla cardíaca.

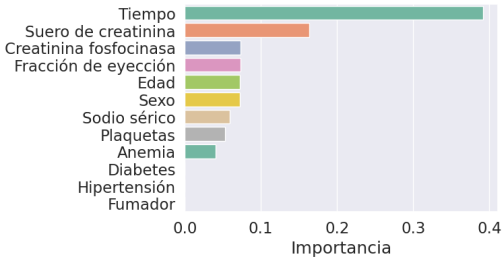

Figura 3. Importancia de las variables predictoras para el modelo de XGBoost

VI. Conclusión

Tanto los modelos de XGBoost como Gradient Boosting exhiben un alto rendimiento en términos de equilibrio entre precisión y exhaustividad, cuando la base de datos contiene tanto variables categóricas como cualitativas.

Se sugiere aplicar regularización al modelo de Gradient Boosting para evitar el sobreajuste y mejorar su rendimiento general. Sin embargo, se prefiere utilizar XGBoost en lugar del Gradient Boosting regular, ya que el proceso de ajuste del modelo con XGBoost es más rápido. XGBoost ofrece una combinación de velocidad y efectividad que lo hace más atractivo en comparación con el Gradient Boosting tradicional.

Referencias

[1] Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*(Vol. 2, pp. 1-758). New York: springer.

[2] Chen, T., & Guestrin, C. (2016, August). *Xgboost: A scalable tree boosting system.In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*(pp. 785-794).