Gradient Boosting y aplicaciones

Victor Manuel Gómez Espinosa
Centro de Investigación en Matemáticas. Unidad Monterrey
Email: victor.gomez@cimat.mx

Abstract – **En este trabajo se describen las ideas principales del modelo GradientBoost, así como las similitudes y diferencias con respecto a la implementación XGBoost la cual actualmente es muy popular. Adicionalmente se implementan en aplicaciones de regresión y clasificación binaria comparándolos contra los modelos regresión lineal y regresión logística respectivamente, demostrando XGBoost resultados sobresalientes en ambas tareas.**

I. INTRODUCCIÓN

Los modelos de aprendizaje maquina se han vuelto muy importantes en la actualidad ya que están presentes en muchos de los elementos de la vida cotidiana, como los sistemas de recomendaciones (compras, películas, música, etc.) o en el correo electrónico ayudarnos a evitar spams, entre otros.

Dentro de estos modelos uno de los mejores actualmente son los de Gradient Boosting por Friedman (2001) quien originalmente planteó la idea del algoritmo para el caso de regresión (la implementación original de este algoritmo fue llamada MART "*Multiple additive regression trees*") y que posteriormente se extendió para el caso de clasificación (Hastie 2009; Izenman 2008). Actualmente la implementación en software libre llamada XGBoost (Chen and Guestrin 2016) es la más relevante, debido a que ganado gran popularidad a raíz de ser la implementación de aprendizaje maquina más utilizada (seguida de Deep neural networks) por la mayoría de los equipos ganadores en diversos concursos (Netflix prize, Kaggle, KDDCup). Algunos de los problemas de estas competencias son: predicción de ventas, predicción del comportamiento de clientes, clasificación de textos, predicción de riesgos, entre otros.

XGBoost cuanta con características muy interesantes que lo han llevado a ser tan popular, algunas de estas son, que aparte de poder realizar regresión, clasificación binaria y multiclase, adicionalmente es capaz de aprender a hacer ranking, es muy rápido, funciona muy bien para muestras muy grandes (billones de observaciones), además de encontrarse disponible libremente para diversas plataformas (C, C++, R, Python, Julia, entre otros).

En este trabajo se busca probar la eficiencia XGBoost para las tareas de regresión y clasificación utilizando datos disponibles en el Machine Learning Repository, compararlos contra otros modelos como el GradientBoost de scikit-learn y los modelos de regresión lineal y logística respectivamente.

II. GRADIENT BOOSTING

En Gradient Boosting el problema de minimización es:

$$\hat{f} = \arg\min_{f} L(f) \tag{1.1}$$

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{1.2}$$

Donde $L(f)$ es la función de costo, y son las observaciones reales y $f(x)$ son las aproximaciones para todo el conjunto de datos.

Algoritmo 1. Gradient Boosting

1. Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$
2. For m=1 to M:
   a. For $i = 1, 2, ..., N$ compute
      $r_{im} = -[g]_{f=f_{m-1}}$
   b. Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, ..., J_m$
   c. For $j = 1, 2, ..., J_m$ compute
      $\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$
   d. Update
      $f_m(x) = f_{m-1}(x) + \eta\left(\sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})\right)$
3. Output $\hat{f}(x) = f_M(x)$

El Algoritmo 1, representa la forma genérica del algoritmo gradient tree-boosting para regresión. De forma general el algoritmo en el paso 1 requiere que se inicialice el primer árbol del modelo (que en este caso es una hoja), minimizando la función de costo respecto a γ que representa el valor de cada hoja en el árbol. Posteriormente, para el paso 2, cíclicamente (a) se obtienen residuales r_{im} con la aproximación inicial, (b y c) con estos residuales se ajusta un árbol y se obtienen sus respectivos valores de sus hojas γ_{jm} (esto se puede realizar mediante el algoritmo CART), (d) se actualiza el modelo mediante la suma del modelo anterior más el nuevo árbol escalado por η (learning rate), se obtienen nuevas predicciones con el nuevo modelo las cuales se utilizarán para los siguientes ciclos hasta el último (M) y finalmente paso 3, se entrega el último modelo. El paso (d) podría decirse que es el que le da el nombre de Gradient Boosting, ya que es justamente lo que hace, irse acercando a la solución por el método del gradiente mediante aproximaciones o pequeños pasos en la dirección correcta, donde el tamaño de paso es η y la dirección está dada por $\sum_{j=1}^{J_m} \gamma_{jm} I\left(x \in R_{jm}\right)$, que corresponde a los valores de la hoja en los que cae cada observación en el árbol actual (γ_{jm} es la predicción de la observación x que cae en la región j del árbol m, y mientras J sea más grande, el árbol es más complejo). Por otro lado, el learning rate al igual que en el método del gradiente su valor es positivo entre 0 - 1, el cual ayuda a no causar un sobre ajuste del modelo (Figuras 1 y 2), y mientras más chico sea más se tardará en llegar a la solución, pero también evitará oscilar alrededor de la solución óptima.

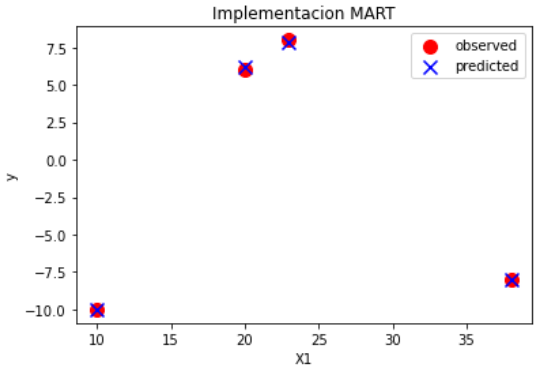

Figura 1: Modelo GradientBoost con sobre ajuste. Learning rate=0.9, arboles=10.

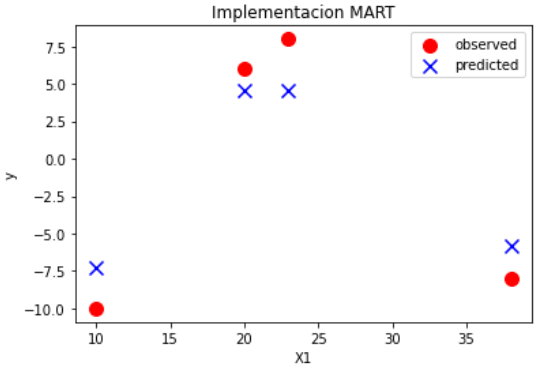

Figura 2: Modelo GradientBoost sin sobre ajuste. Learning rate=0.3, arboles=10.

Otra observación importante de este método es que a diferencia de otros como por ejemplo AdaBoost que utiliza arboles de un solo nivel (stumps), Gradient Boosting el primer árbol es una hoja, pero los demás pueden ser stumps o arboles de diferente nivel (Figura 3).

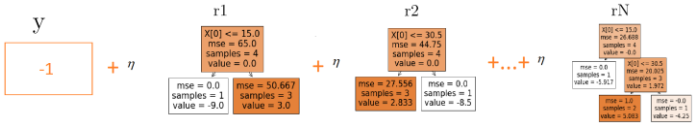

Figura 3: Esquema de un modelo Gradient Boosting.

Para el paso 1 donde se requiere minimizar la función de costo L (1.2) respecto a γ, para resolver esto se utiliza una aproximación por serie de Taylor de segundo orden:

$$L^{(m)} \simeq \sum_{i=1}^{N}\left[l\left(y_i, \hat{y}^{(m-1)}\right) + g_i f_m\left(x_i\right) + \frac{1}{2} h_i f_m^2\left(x_i\right)\right] \quad (1.3)$$

Donde $l\left(y_i, \hat{y}^{(m-1)}\right)$ es una constante, g_i el gradiente y h_i el hessiano. Sustituimos $f_m\left(x_i\right)$ por γ_{jm} y derivamos respecto γ_{jm} e igualamos a 0 para minimizar:

$$\partial L \Big/ \partial \gamma = \sum_{i=1}^{N} g_i + \sum_{i=1}^{N} h_i \gamma_{jm} = 0 \quad (1.4)$$

Despejando γ:

$$\gamma_{jm} = \frac{\sum_{i=1}^{N} -g_i}{\sum_{i=1}^{N} h_i} \quad (1.5)$$

Para el caso de regresión la función de costo empleada es:

$$L(f) = \frac{1}{2}\left[y_i - f(x_i)\right]^2 \tag{1.6}$$

Y su correspondiente gradiente g y hessiano h son:

$$g_i = -\left(y_i - f(x_i)\right) \tag{1.7}$$

$$h_i = 1 \tag{1.8}$$

Donde se puede observar que el negativo del gradiente corresponde al residual para el árbol m evaluado en las aproximaciones del ciclo anterior: $r_{im} = -[g]_{f=f_{m-1}}$.

Entonces sustituyendo (1.7) y (1.8) en (1.5) tenemos:

$$\gamma_{1,0} = \frac{\sum_{i=1}^{N}(y_i)}{N} \tag{1.9}$$

Que es equivalente al promedio de los residuales en cada hoja del árbol, donde para el primer árbol (m=0), no tenemos predicciones $(f(x)=0)$, sólo las observaciones y, por lo tanto, es el promedio de las observaciones. Para el paso 2c se realiza lo mismo, notando que la parte izquierda del término $f_{m-1}(x_i)+\gamma$ hace referencia a las predicciones del paso anterior y sustituyendo esto nos daría los residuales, y, por lo tanto, el promedio de los residuales en cada hoja del árbol.

$$\gamma_{jm} = \frac{\sum_{i\in I_j} -g_i}{\sum_{i\in I_j} h_i} \tag{1.10}$$

Se pueden utilizar otras funciones de costo y seguir el mismo procedimiento, por ejemplo, para el caso de clasificación binaria utilizar la función de costo logística:

$$L(f) = y_i \ln\left(1+e^{-f(x_i)}\right) + (1-y_i)\ln\left(1+e^{f(x_i)}\right) \tag{1.11}$$

III. XGBoost

Algunas de las características que hacen diferente a XGBoost del modelo original (GradientBoost) es que introduce un término de regularización $\Omega(f_m)$ en la función de costo (1.2) convirtiéndola en:

$$L(f_m) = \sum_{i=1}^{N} L(y_i, f_m(x_i)) + \Omega(f_m) \tag{1.12}$$

Donde:

$$\Omega(f_m) = \frac{1}{2}\lambda\sum_{j=1}^{J}\gamma_{jm} + \tau J \tag{1.13}$$

El cual transforma la ecuación (1.10) en:

$$\gamma_{jm} = \frac{\sum_{i\in I_j} -g_i}{\sum_{i\in I_j} h_i + \lambda} \tag{1.14}$$

Los parámetros λ y τ son valores positivos mayores a cero, donde por ejemplo si $\lambda > 0$ en (1.14), va a disminuir el valor de salida γ_{jm} que tiene cada región en el árbol (cada hoja), y por lo tanto ayuda a evitar el problema del sobre ajuste del modelo.

Sustituyendo (1.14) en la función de costo obtenemos el correspondiente optimo:

$$L(\gamma_{jm}) = -\frac{1}{2}\sum_{j=1}^{J}\frac{\left(\sum_{i\in I_j} -g_i\right)^2}{\left(\sum_{i\in I_j} h_i\right)+\lambda} + \tau J \tag{1.15}$$

esta ecuación nos da una puntuación que podemos utilizar para cuantificar la calidad de la estructura de un árbol y mientras menor sea la puntuación mejor es la estructura (similar al modelo CART con el índice Gini), por lo tanto, la forma de construir arboles es diferente. Para construir el árbol se comienza por una hoja e iterativamente se van añadiendo ramas izquierda (I_L) y derecha (I_R) y para cada división se calcula la puntuación total con la ecuación (1.16) donde el parámetro τ es para regular el número de hojas del árbol y representa el costo de introducir hojas adicionales (Figuras 4 y 5), es decir que si obtenemos un resultado negativo termina de hacer divisiones y recursivamente quita las hojas que obtuvieron puntuaciones negativas (Chen 2014).

$$L_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} -g_i\right)^2}{\left(\sum_{i\in I_L} h_i\right)+\lambda} + \frac{\left(\sum_{i\in I_R} -g_i\right)^2}{\left(\sum_{i\in I_R} h_i\right)+\lambda} - \frac{\left(\sum_{i\in I} -g_i\right)^2}{\left(\sum_{i\in I} h_i\right)+\lambda}\right] - \tau \tag{1.16}$$

Otra de las diferencias con GradientBoosting en la construcción de los árboles es que tiene la opción de que en lugar de buscar entre todos los puntos posibles de división lo puede hacer de manera más eficiente (sobre todo para situaciones donde la cantidad de datos sobrepasa la cantidad de memoria) proponiendo puntos para hacer las divisiones de acuerdo a percentiles de la distribución de las variables.

Figura 4: Árbol complejo de 5 niveles. $\tau = 0, \lambda = 0$

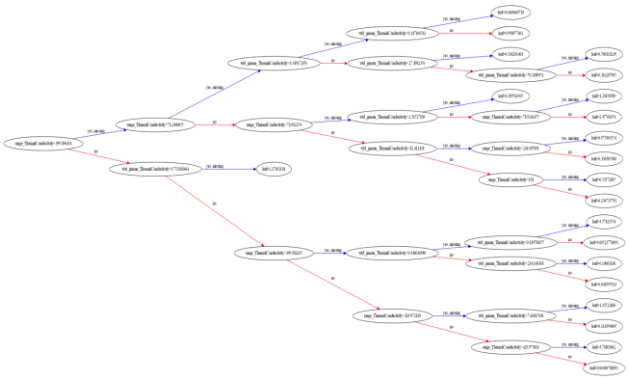

Figura 5: Árbol de 5 niveles de menor complejidad que el de la Figura 4. $\tau = 50, \lambda = 50$

Entre otras de las características interesantes de XGBoost es que para las observaciones que tienen datos faltantes en alguna variable o ceros, por ejemplo, propone una dirección predeterminada en los árboles que aprende de los datos, la cual ayuda a que algoritmo se ejecute hasta 50 veces más rápido que sin esta característica. Además, tiene otras características que ayudan a optimizar como se utilizan los recursos de la maquina como por ejemplo guarda los datos en bloques de memoria, partes del código que son paralelizadas, además de hacer uso de la memoria cache dentro del CPU (la cual es la más rápida).

IV. Implementación y resultados

Se utilizaron datos disponibles en el Machine Learning Repository par los casos de regresión y clasificación. Para el caso de regresión particularmente se intentó reproducir los resultados del trabajo de (Hamidieh 2018) para reproducir la temperatura critica de materiales superconductores. El conjunto de datos cuenta con 80 variables numéricas y 21263 observaciones. Se utilizaron las métricas de la raíz del error cuadrático medio (RMSE), R2, y el tiempo de ejecución y un sistema de k-fold con 5 folds para seleccionar los mejores modelos (Figura 7). Los modelos a comparar fueron regresión lineal, GradientBoost y XGBoost (Tabla 1 y figura 6).

	T	RMSE	R2
LR	01.6s	17.65	0.74
GB	16min 23s	10.21	0.91
XGB	10min 15s	8.98	0.93

Tabla 1: Resultados para el caso de regresión.

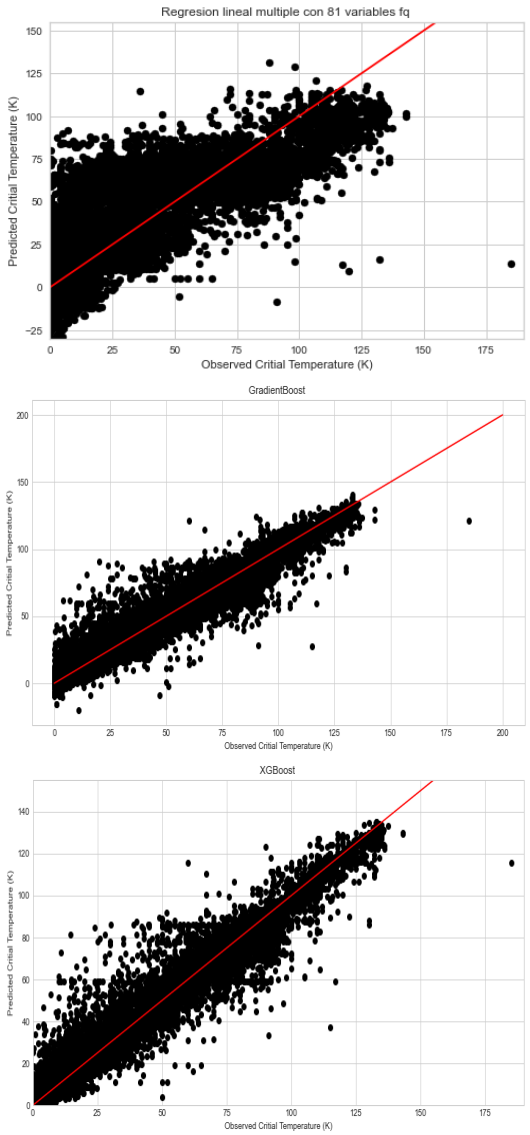

Figura 6: Resultados para los modelos: Regresión lineal, GradientBoost, XGBoost. Mientras mas alineados se encuentren a la línea roja, el modelo es mejor.

Otra característica de XGBoost es que permite conocer las variables mas importantes dentro del modelo

(Figuras 8 y 12, para los modelos de regresión y clasificación respectivamente).

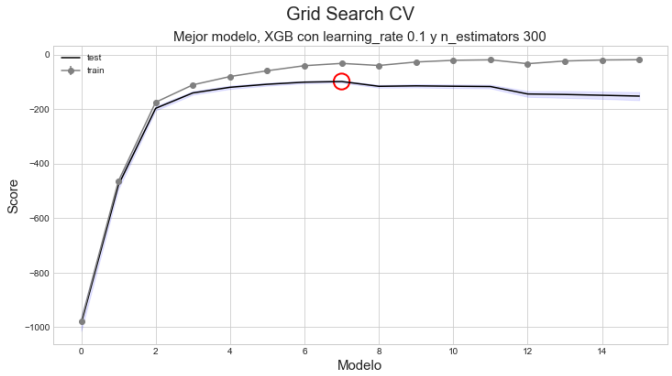

Figura 7: Gráfico del desempeño de los modelos que se probaron para XGBoost y las características del mejor modelo.

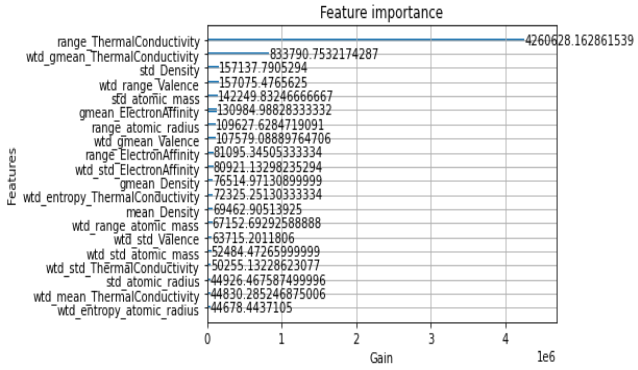

Figura 8: Variables más importantes según el Gain dentro del mejor modelo XGBoost para regresión.

Para el caso de clasificación se utilizaron los datos de una campaña de marketing de un banco (Moro et al. 2011) particularmente se intentó predecir si los clientes contratarían un producto del banco. El conjunto de datos cuenta con 16 variables numéricas y categóricas con 45211 observaciones además de ser un conjunto de datos desbalanceado (Figura 9). Se utilizaron las métricas del f1 score, AUC score y el tiempo de ejecución además de un sistema de k-fold de 5 folds para seleccionar los mejores modelos (Figura 11). Los modelos a comparar fueron regresión logística, GradientBoost y XGBoost (Tabla 2 y figura 10).

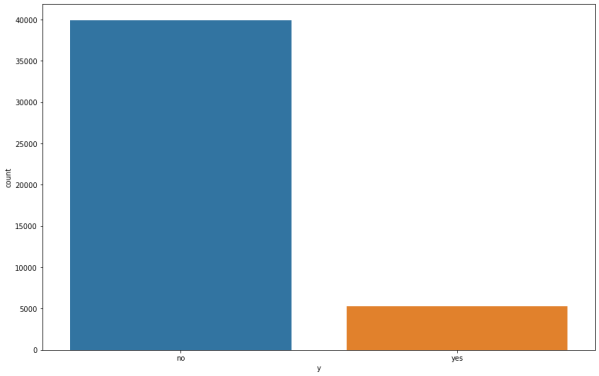

Figura 9: Conjunto de datos del banco desbalanceado respecto a la variable respuesta con la característica de interés 'yes'.

	T	F1	AUC
LOG R	3min 02.7s	0.02	0.579
GB	7min 14s	0.49	0.902
XGB	9min 16s	0.51	0.904

Tabla 2: Resultados para el caso de clasificación.

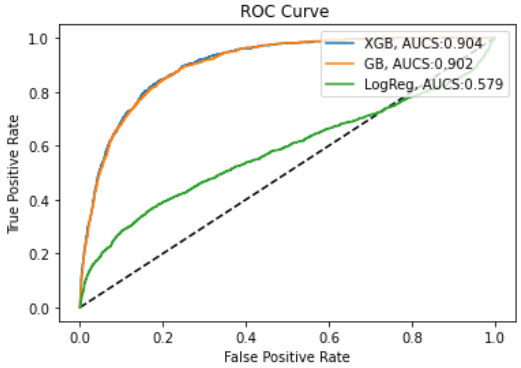

Figura 10: Gráfico ROC comparativo de los modelos, con los valores del AUC score para cada modelo.

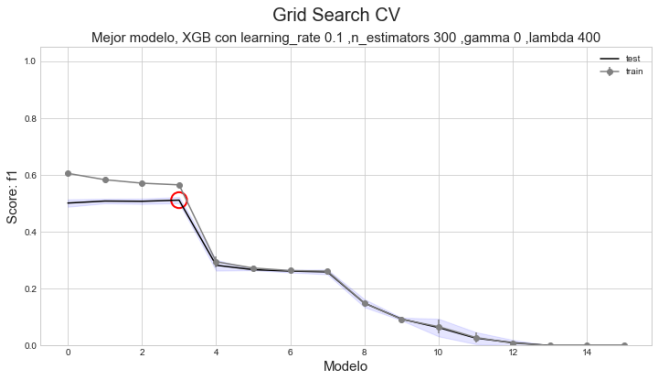

Figura 11: Gráfico del desempeño de los modelos que se probaron para XGBoost y las características del mejor modelo.

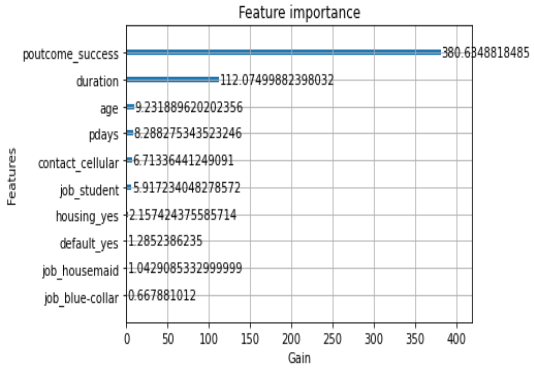

Figura 12: Variables más importantes según el Gain dentro del mejor modelo XGBoost para clasificación.

IX. Conclusiones

XGBoost fue el mejor modelo tanto como para regresión como para clasificación, dejando lejos a los modelos de regresión lineal y logística respectivamente. Esto se puede explicar debido a la característica de los modelos GradientBoost que utilizan árboles, y la forma en la que los construyen (particularmente XGBoost), puesto que permite captar mejor las características importantes en las variables.

XGBoost resultó el modelo más rápido para el caso de regresión, mientras que GradientBoost resultó el más rápido para el caso de clasificación, esto posiblemente debido a que en XGBoost se variaron los parámetros de regularización τ, λ , los cuales tienen un efecto en evitar la complejidad de los árboles podándolos recursivamente y esto podría tomar mayor tiempo computacional que solo restringir un tamaño máximo como en el casi de GradientBoost.

Como trabajo futuro se podrían comparar casos con mas observaciones y datos faltantes para probar como funciona XGBoost o probar la clasificación multiclase o ranking. También probar estos mismos datos con otros modelos como redes neuronales para tener otra perspectiva de comparación, además de mejorar el modelo de clasificación mediante pesos para la característica de interés.

References

Chen, T. (2014), "Introduction to Boosted Trees," *Data Mining with Decision Trees*, 187–213. https://doi.org/10.1142/9789812771728_0012.

Chen, T., and Guestrin, C. (2016), "XGBoost: A scalable tree boosting system," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-Augu, 785–794. https://doi.org/10.1145/2939672.2939785.

Hamidieh, K. (2018), "A data-driven statistical model for predicting the critical temperature of a superconductor," *Computational Materials Science*, 154, 346–354. https://doi.org/10.1016/j.commatsci.2018.07.052.

Hastie, T. et. all. (2009), "Springer Series in Statistics The Elements of Statistical Learning," *The Mathematical Intelligencer*, 27, 83–85. https://doi.org/10.1007/b94608.

Izenman, A. J. (2008), *Springer Texts in Statistics Alan Julian Izenman Regression , Classification , and*.

Moro, S., Laureano, R. M. S., and Cortez, P. (2011), "Using data mining for bank direct marketing: An application of the CRISP-DM methodology," *ESM 2011 - 2011 European Simulation and Modelling Conference: Modelling and Simulation 2011*, 117–121.