UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

UN MODELO PREDICTIVO INTERPRETABLE PARA LA ESTIMACIÓN DEL INGRESO MONETARIO DE CLIENTES BANCARIOS BASADO EN XGBOOST Y SHAP.

POR

Vicente Manuel Marchant Contreras

Tesis presentada a la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Matemático.

Profesores Guía: Dr(c). Patricio Salas Fernández y Dr. Guillermo Ferreira Cabezas

Octubre 2022

Concepción, Chile

A mi familia.

AGRADECIMIENTOS

Quiero aprovechar esta instancia para mostrar mi gratitud a las personas que hicieron mi paso por la universidad una experiencia inolvidable.

Agradezco en primer lugar a los/as docentes y funcionarios/as de los Departamentos de Ingeniería Matemática y de Estadística de la Universidad de Concepción con los que tuve la oportunidad de aprender y compartir.

A mis profesores guía, Patricio Salas, por su buena disposición y acompañamiento durante este proceso y por darme la libertad de abordar temas de mi interés. A Guillermo Ferreira, por su buena disposición y por tomar en cuenta mis -quizás apresurados- acercamientos para empezar mi tesis. A Pamela Meléndez, por toda su comprensión y por darme la oportunidad de compartir con un gran equipo de trabajo. A la profesora Mónica Selva, por su gran guía como Jefa de Carrera durante mis años universitarios.

A mi padre Ricardo, por sus consejos y apoyo incondicional. A mi hermano Pablo, por ser un gran compañero de vida. A mi madre Erika, que a pesar de no estar aquí, sé que estaría orgullosa por mi. A Camila, por los momentos lindos que hemos vivido y por los que vendrán, por todo el amor que compartimos, sé que seguiremos nutriendo nuestra relación y seremos profesionales íntegros.

A Javiera, por ser un gran apoyo y una gran amiga durante estos largos años de carrera. A Alberto, por su amistad y por las innumerables idas a tomar café. A Felipe, Tomás, Pablo e Ignacio, por los años de amistad y por acompañarnos a pesar de la distancia. A Patricio y Arthur, por su compañía en los primeros años de universidad. A Claudio, por todo su apoyo. A Fernanda, por el cariño compartido.

Finalmente pero no menos importante, agradezco al profesor Oscar Sánchez, que motivó en mí el interés por las matemáticas desde la enseñanza media.

Resumen

Usualmente, las instituciones bancarias no cuentan con información actualizada de la renta o ingresos mensuales que reciben sus clientes. Esta información es utilizada para mejorar la gestión de oferta de productos, como por ejemplo, segmentar a los clientes y ofrecer tipos de productos diferenciados. Por otra parte, estas entidades construyen modelos predictivos donde pueden llegar a utilizar cientos de variables explicativas, de las cuales solo un subconjunto de estas realmente contribuyen en capturar la variabilidad de la respuesta. En este trabajo se propone una metodología que permite entrenar un modelo de Machine Learning (XGBoost) con un subconjunto reducido de variables explicativas, en comparación a la base completa de variables que utiliza la institución. Esto, bajo el supuesto de que tener el número óptimo de variables explicativas puede igualar o aumentar el poder predictivo y disminuir la complejidad del modelo. Luego, para transparentar las predicciones obtenidas por el modelo, se ofrece un análisis de interpretabilidad utilizando el método Shapley Additive Explanations (SHAP) Lundberg and Lee (2017). Para realizar la reducción del número de variables explicativas se compararon y aplicaron dos métodos de selección de variables, Boruta-XGBoost (Kursa and Rudnicki (2010), Alsahaf et al. (2022)) y BorutaShap Keany (2020). La metodología propuesta se testeó sobre datos simulados, en donde las variables explicativas creadas tuvieron asociados distintos pesos. El objetivo principal del estudio de simulación fue ver si los métodos eran capaces de seleccionar como "importantes" aquellas variables que dependían directamente de la respuesta (de la forma $Y = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$), y como "no importantes" o "no informativas" a aquellas que a priori no estaban relacionadas con la respuesta ($x_{p+1}, \ldots, x_{p+q}$, con p, q fijos). Finalmente, aplicamos nuestra metodología sobre un conjunto de datos real. Este cuenta con los registros de renta de 10.000 clientes bancarios y un total de 426 variables explicativas. Los resultados muestran que el método BorutaShap ofrece un subconjunto de 35 variables que aumentan el poder predictivo del modelo XGBoost, superando incluso al modelo entrenado con las 426 variables originales en cuanto a porcentaje de éxito. Este trabajo representa un aporte para las instituciones financieras, ya que a partir de la adopción de nuestra metodología serán capaces de identificar de forma consistente y dar seguimiento a las variables más influyentes, pudiendo además reducir la complejidad y el tiempo de entrenamiento de los modelos XGBoost sin sacrificar el poder predictivo de los mismos.

Palabras Clave – Extreme Gradient Boosting (XGBoost), métodos de selección de variables, Shapley Additive Explanations (SHAP), renta de clientes bancarios.

Abstract

Usually, banking institutions do not have up-to-date information on the monthly income received by their clients. This information is useful for improving the management of product offerings, for example, segmenting clients and offering differentiated types of products. On the other hand, financial institutions build predictive models where they can use hundreds of features, of which only a subset of these contribute to capturing the variability of the response. In this work we propose a methodology for training a Machine Learning model (XGBoost) with a subset of explanatory features, in comparison with the complete set of features used by the institution. This, assuming that having the optimal number of explanatory features can equal or increase the predictive power and decrease the complexity of the model. Then, in order to make the predictions obtained by the model transparent, an interpretability analysis was performed using the Shapley Additive Explanations (SHAP) (Lundberg and Lee, 2017) method. To perform the reduction of the number of explanatory features, two feature selection methods were compared and applied. Boruta-XGBoost (Kursa and Rudnicki, 2010; Alsahaf et al., 2022) and BorutaShap (Keany, 2020). First, a simulation study was performed; In this study, the explanatory features had different weights associated with them. The main objective of the simulation study was to see if the methods were able to select as "important" those features that depended directly on the response (of the form $Y = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$), and as "unimportant" or "non-informative" those that a priori were not related to the response ($x_{p+1}, \ldots, x_{p+q}$, with p, q fixed). Finally, we applied our methodology to a real dataset. This dataset includes the income records of 10.000 bancary clients and 426 explanatory features. The results shows that the BorutaShap method offers a subset of 35 features that increase the predictive performance of the XGBoost model in terms of success rate, surpassing even the original model trained with 426 features. This work represents a contribution to financial institutions, since, by adopting our methodology they will be able to consistently identify and follow up on the most influential variables, reducing the complexity and training time of the XGBoost models without sacrificing their predictive power.

Keywords – Extreme Gradient Boosting (XGBoost), feature selection methods, Shapley Additive Explanations (SHAP), bancary clients income.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Las entidades financieras se enfrentan a la incertidumbre de que muchos de sus clientes no tienen información actualizada sobre los ingresos monetarios totales que estos reciben. Los ingresos monetarios pueden entenderse como la suma de todas las ganancias mensuales tales como sueldos, pensiones, renta por arriendos, entre otros. La información de los ingresos monetarios es utilizada por las instituciones financieras en una amplia variedad de aspectos claves para la gestión interna y para el contacto comercial con los clientes, algunos aspectos relevantes son:

- Mejorar la asignación de ofertas de distintos productos bancarios.
- Ofrecer montos de crédito óptimos, es decir, ofrecer más cupo a los clientes de mayor renta, y menor cupo a los de menores rentas, con el fin de evitar el sobre-endeudamiento.
- Segmentar a los clientes existentes según sus ganancias, para así ofrecer tipos de productos diferenciados.

Tal como mencionamos anteriormente, todos los puntos recién nombrados forman parte importante de la gestión interna de las instituciones financieras, por lo que se hace útil disponer de una estimación lo más precisa posible de los ingresos monetarios de los clientes a partir de datos provenientes de diversas fuentes de información (e.j., nivel educacional, lugar de residencia, situación laboral, etc.).

En los últimos años, las instituciones financieras han adoptado nuevas metodologías basadas en el uso de técnicas de Machine Learning (ML) como complemento a los modelos estadísticos comúnmente utilizados. En la práctica, los modelos predictivos pueden ser entrenados con cientos de variables explicativas, de las cuales solo un subconjunto de ellas realmente contribuyen en capturar el comportamiento de la variable objetivo. Por otra parte, el uso de modelos de ML más complejos como los basados en árboles o redes neuronales, traen consigo una pérdida en la interpretabilidad (Molnar, 2022). De esto, surge el interés de

desarrollar una metodología que permita seleccionar las variables más importantes y por otra parte darle interpretabilidad a las predicciones del modelo. A continuación, se dará un contexto general del problema a abordar junto con algunos conceptos previos.

El ML es el estudio de algoritmos capaces de mejorar o aprender automáticamente a través de la experiencia y el uso de datos (Mitchell, 1997). Los enfoques más comunes del aprendizaje automático son el supervisado y el no supervisado. En el aprendizaje supervisado, el algoritmo es capaz de generalizar o aprender en base a datos con etiqueta, ya sea para problemas de clasificación o regresión (Hastie et al., 2001). Por otro lado, el aprendizaje no supervisado intenta extraer características y patrones desde datos sin etiquetar (Hastie et al., 2001). Las técnicas de ML se han destacado por su eficiencia computacional y por lograr resultados de predicción superior a los obtenidos con las técnicas estadísticas comúnmente utilizadas (e.j. Shin et al., 2021; Lang et al., 2022). Actualmente, el ML es ocupado en una variedad de aplicaciones, tales como finanzas, robótica, reconocimiento de patrones, procesamiento de lenguaje natural, diagnósticos médicos, algoritmos de recomendación para redes sociales, predicción del modo de viaje de personas, entre muchas otras (e.j. Ray, 2019; Salas et al., 2022).

A pesar de las ventajas predictivas de los algoritmos de ML, en la comunidad científica estos métodos son criticados por su naturaleza de "caja-negra" (Salas et al., 2022), y esta es la principal barrera para adoptar estos métodos en otras áreas (Molnar, 2022). Además, según Doshi-Velez and Kim (2017), el problema es que una sola métrica, como la precisión de la predicción, es una descripción incompleta de la mayoría de las tareas del mundo real.

De acuerdo con Molnar (2022), al momento de entrenar un modelo predictivo podría no importar el por qué se tomó una decisión, basta con saber que el rendimiento en un conjunto de datos de prueba fue bueno. Pero en otros casos, conocer el "por qué" puede ayudar a entender mejor el problema, los datos y la razón por la cual un modelo puede fallar. Por esta razón, se hace necesario incorporar un análisis de interpretabilidad de las predicciones para generar confianza en los resultados del modelo (Molnar, 2022; Salas et al., 2022).

Según Miller (2017), la interpretabilidad es el grado en que una persona puede entender la causa de una decisión. En esta linea, Gunning et al. (2019) define la interpretabilidad de un modelo como la medida en que las salidas son explicadas mediante las variables de entrada. Cuanto mayor sea la interpretabilidad de un modelo de ML, más fácil será para alguien comprender por qué el modelo ha tomado determinadas decisiones o llegado a ciertos resultados (Molnar, 2022). Esta medida o valores de importancia pueden calcularse para una sola predicción (individualizada) como para todo el conjunto de datos y así explicar

el comportamiento general de un modelo (global) (Lundberg et al., 2019). Dentro de los métodos comunes de interpretabilidad global y local destacan; Gráficos de dependencia parcial, *permutation feature importance*, *local surrogate models* (LIME), Shapley Additive Explanations (SHAP), entre otros (más información, revisar Molnar (2022)).

En Lundberg et al. (2019) se expone que los métodos comunes de interpretabilidad para modelos basados en árboles como los entregados por XGBoost (Chen and Guestrin (2016)), *sklearn* (Pedregosa et al. (2011)) y la librería *gbm* de R (Ridgeway (2005)) son inconsistentes, debido a que pueden reducir la importancia asignada a una variable cuando el verdadero impacto de esa variable en realidad aumenta. Por esta razón, en este trabajo de tesis se utilizará la metodología SHAP (Shapley Additive Explanations), desarrollada por (Lundberg and Lee, 2017), la cual está basada en conceptos de teoría de juegos cooperativos, siendo los valores Shapley (Shapley, 1953) los únicos que cumplen axiomas deseables para distribuir una "ganancia" o valores de importancia de forma consistente a cada variable del modelo (Molnar, 2022).

Algunas ventajas del método SHAP son:

- Tiene una base matemática sólida basada en teoría de juegos cooperativos.
- Dispone de una implementación rápida para modelos basados en árboles.
- El cálculo rápido permite computar los numerosos valores de Shapley necesarios para las interpretaciones globales del modelo.

El objetivo de este trabajo de tesis es medir la capacidad predictiva del modelo Extreme Gradient Boosting (XGBoost (Chen and Guestrin, 2016)), comparar y aplicar dos métodos de selección de variables y finalmente ofrecer un análisis de la interpretabilidad local y global de las predicciones del modelo mediante Shapley Additive Explanations (SHAP), tal como lo han realizado otros autores (e.j. Feng et al., 2021; Parsa et al., 2020; Yang et al., 2021; Chelgani, 2021).

1.1. Revisión de Literatura

La estimación del ingreso monetario de clientes bancarios es un tema que ha sido poco estudiado en la literatura, debido a temas de confidencialidad de la información de este tipo de instituciones. De acuerdo a la revisión, el único trabajo encontrado respecto al tema es el de Kibekbaev and Duman (2016), en este trabajo predicen los ingresos de los clientes bancarios analizando el rendimiento de varios modelos de regresión.

A continuación, presentamos evidencia bibliográfica que muestra el amplio uso del modelo XGBoost en distintas áreas del conocimiento. Además, se detallan algunos trabajos que se centran en la idea de generar modelos interpretables basados en la combinación del modelo XGBoost y el método SHAP. Por último, mostramos trabajos que han usado métodos de selección de variables explicativas y en particular que hayan utilizado la idea detrás del método SHAP para este propósito.

- En Kibekbaev and Duman (2016), se comparan 16 algoritmos de regresión lineal y no lineal para la predicción de ingresos de clientes bancarios de diversos bancos en Turquía. Se obtuvieron mejores resultados en algoritmos de regresión no lineal en comparación con algoritmos de regresión lineal. Se tomaron en cuenta métricas tales como error cuadrático medio, error medio absoluto, R^2, etc.

- Dwidarma et al. (2021), compara entre regresión logística y XGBoost para predecir si un cliente bancario será un buen pagador o no (modelo de clasificación para variable binaria). Se obtuvieron mejores resultados con el algoritmo XGBoost (exactitud $99{,}94\,\%$, sensitividad $99{,}88\,\%$, especificidad $99{,}99\,\%$ y precisión $99{,}99\,\%$), en comparación a la regresión logística (exactitud $87{,}88\,\%$, sensitividad $82{,}43\,\%$, especificidad $93{,}13\,\%$ y precisión $92{,}05\,\%$).

- Husna et al. (2020) compara los modelos XGBoost, Maquinas de Soporte Vectorial (SVM) y Redes Neuronales Artificiales (ANN), con el objetivo de predecir una variable continua relacionada al diseño de un medicamento utilizado en el tratamiento de la Diabetes Mellitus. Por amplio margen se obtuvieron mejores resultados para el modelo XGBoost.

- Feng et al. (2021) entrenó un modelo explicativo basado en XGBoost y SHAP para predecir la resistencia de un muro cortante de hormigón armado, obteniendo una exactitud de $97\,\%$ para el conjunto de validación, siendo mejor que los modelos semiempíricos comunes para la mecánica de solidos. Finalmente, se hizo un análisis para estimar la importancia relativa de las variables y así identificar atributos clave al momento de estimar la resistencia del muro.

- En Wang and Ross (2018) emplean un modelo XGBoost y un modelo de elección discreta para predecir la elección del modo de viaje. En esta misma linea, en Salas et al. (2022) realizan una comparación sistemática entre modelos de elección discreta y cinco algoritmos de ML, incluyendo XGBoost, para modelar el mismo fenómeno.

- Marcílio and Eler (2020) demuestra la utilidad de utilizar el método SHAP para una selección óptima de variables, además del uso común de explicar las predicciones de un modelo. Se muestra que SHAP obtiene mejores resultados que otros algoritmos de

selección de variables.

1.2. Objetivos

El objetivo de este trabajo es entrenar un modelo Extreme Gradient Boosting (XGBoost) para estimar ingresos mensuales de clientes bancarios (variable continua) con un acotado número de variables explicativas. Para esto, se realizará una selección de variables explicativas, comparando entre dos métodos comúnmente utilizados en la literatura de ML, a saber; Boruta-XGBoost y BorutaShap. Luego, una vez determinado el mejor subconjunto de variables explicativas se entrenará un modelo XGBoost, a partir del cual realizaremos un análisis de interpretabilidad global y local de las predicciones mediante el método SHAP. Esto, con el fin de identificar variables y grupos de variables que más influyen en el ingreso monetario de los clientes. Los datos utilizados los provee una institución bancaria Chilena anónima, y están conformados por 10.000 registros de ingresos de clientes bancarios junto con 426 variables explicativas. Los modelos y métodos de esta tesis serán desarrollados mediante Python 3.9 (Van Rossum and Drake, 2009).

1.2.1. Objetivos específicos

- Entrenar modelos XGBoost para predecir el ingreso monetario de clientes bancarios.
- Comparar y aplicar métodos de selección de variables mediante un estudio de simulación, con el fin de eliminar aquellas variables que tienen contribuciones bajas o nulas sobre el desempeño predictivo del modelo.
- Analizar la interpretabilidad global y local de las predicciones del modelo final mediante el uso del método SHAP.

Capítulo 2

Marco Teórico

En este capítulo se presentan las principales definiciones y conceptos previos de los métodos utilizados en el desarrollo de este proyecto de título, partiendo desde el funcionamiento de los modelos basados en árboles; Gradient Boosting y Extreme Gradient Boosting, pasando por la teoría de juegos cooperativos y valor de Shapley, para luego definir los problemas de selección de variables y búsqueda de hiperparámetros. Finalmente, se presentan las métricas de evaluación y comparación de los modelos entrenados.

2.1. Modelos basados en árboles

En términos generales, los modelos basados en árboles dividen el espacio de variables en un conjunto de rectángulos y ajustan un modelo sencillo en cada uno de ellos. Estos modelos se pueden aplicar tanto para problemas de regresión como de clasificación. Nos enfocaremos en los árboles de regresión debido a la naturaleza del problema a abordar.

Inicialmente, en Leo Breiman (1984) se define el concepto de *Bagging*, que consiste en un método que combina las predicciones de varias secuencias de datos para crear un resultado más preciso. Esto puede ser aplicado a árboles de decisión, separando un conjunto de datos en N partes, ajustando un árbol de decisión a cada uno y luego promediando las salidas para obtener un resultado final.

Luego de esto surgió el concepto de *Boosting*, que sigue una lógica similar al método Bagging. Se separa un conjunto de datos en N partes, donde los predictores (en nuestro caso, árboles de decisión) aprenden de los errores que han cometido los árboles anteriores, obteniendo el resultado final a través del último árbol.

Con esto, se define un *modelo de ensamble* como aquel que está formado por un conjunto de árboles de decisión individuales, los cuales son entrenados de forma secuencial, de forma de que cada nuevo árbol utiliza la información del árbol anterior para aprender de sus errores. La predicción de una nueva observación se obtiene sumando las predicciones de todos los árboles que componen al modelo. Esta es la lógica que siguen los modelos Gradient Boosting y Extreme Gradient Boosting.

2.1.1. Gradient Boosting

Siguiendo con los modelos aditivos de árboles, definiremos el modelo Gradient Boosting. Inicialmente, un árbol de decisión particiona el espacio de variables en regiones disjuntas R_j, $j = 1, \ldots, J$. Estas regiones representan los J nodos u hojas terminales del árbol. Una constante b_j se le asigna a cada región, tal que

$$x \in R_j \Longrightarrow f(x) = b_j.$$

Así, podemos definir un árbol como

$$h_m(x) := \sum_{j=1}^{J} b_j I(x \in R_j).$$

El algoritmo Gradient Boosting busca una aproximación $\hat{F}(x)$ como una suma ponderada de M árboles de decisión $h_m(x)$, denominados aprendices débiles,

$$\hat{F}(x) = \sum_{m=1}^{M} \gamma_m h_m(x).$$

Dado un set de n datos de entrenamiento, el objetivo del método es minimizar el valor de una función de pérdida L. La forma de hacer esto consiste en partir de un valor $F_0(x)$ constante e ir incrementándolo. Así, la predicción del modelo en el m-ésimo paso se obtiene mediante

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x),$$

donde

$$\gamma_m = \arg\min_{\gamma} \sum_{x=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

A continuación, se describe el funcionamiento del algoritmo Gradient Boosting:

(1) Como input se recibe un conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^{n}$, una función diferenciable $L(y, F(x))$ y un número M de iteraciones.

(2) Para cada $m = 1$ hasta M se calculan los pseudo-residuos dados por

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}},$$

para $i = 1, \ldots, n$. Luego, calcular el resultado de un árbol $h_m(x)$ utilizando los pseudo-residuos, es decir, el conjunto $\{(x_i, r_{im})\}_{i=1}^{n}$, de esta forma, el modelo aprende de los errores del paso anterior.

(3) Se calcula γ_m al resolver el siguiente problema de optimización

$$\gamma_m = \arg\min_{\gamma} \sum_{x=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

(4) Se actualiza el modelo

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

De esta forma, se obtiene la predicción final $F_M(x)$ en el paso M.

2.1.2. Extreme Gradient Boosting (XGBoost)

El algoritmo Extreme Gradient Boosting (XGBoost) surge como una implementación eficiente del método Gradient Boosting, desarrollada por Chen and Guestrin (2016). Entre las principales características se destaca el incluir términos de penalización para evitar el sobreajuste, reducción proporcional de las hojas de los árboles, método de Newton-Raphson para función de pérdida e implementación eficiente para entrenamiento en multiprocesadores.

Se define la siguiente función objetivo, la cuál queremos minimizar, a modo de ayudar al aprendizaje del modelo

$$\mathcal{L}(\phi) = \sum_i L(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \tag{2.1.1}$$

donde L es una función diferenciable que mide la diferencia entre la predicción $\hat{y}_i$ y el valor real y_i, y Ω se define como

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||_{L^2},$$

donde T es el número de hojas en el árbol, $w \in \mathbb{R}^T$ los pesos de las hojas del árbol y los términos de regularización γ y λ ayudan a evitar el sobreajuste. En términos generales, Ω penaliza la complejidad del modelo, esta es una de las características que diferencia al Gradient Boosting del Extreme Gradient Boosting.

Por otro lado, se utiliza una aproximación de segundo orden para la función de pérdida. Primero, se define $\hat{y_i}^{(t)}$ como la predicción de la instancia i en la iteración t, agregaremos f_t definido anteriormente a la función objetivo (2.1.1).

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)) + \Omega(f_t). \tag{2.1.2}$$

Luego, la aproximación de segundo orden a f:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [L(y_i, \hat{y_i}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t), \tag{2.1.3}$$

donde $g_i = \partial_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$ y $h_i = \partial^2_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$ son los gradientes de primer y segundo orden de la función L.

A continuación, se describe el funcionamiento básico del algoritmo XGBoost:

(1) Como input se recibe un conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^n$, una función de pérdida al menos dos veces diferenciable, un número M de iteraciones y $\alpha \in \mathbb{R}^+$ una tasa de aprendizaje.

(2) Se inicia el algoritmo con un valor constante

$$f_0(x) = \arg\min_{\theta} \sum_{i=1}^{n} L(y_i, \theta).$$

(3) Para $m = 1, \ldots, M$ y para $i = 1, \ldots, n$ se calculan los gradientes de primer y segundo orden

$$g_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{m-1}(x)},$$
$$h_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial^2 f(x_i)}\right]_{f(x)=\hat{f}_{m-1}(x)}.$$

(4) Luego, se calcula el resultado de un árbol utilizando el conjunto $\left\{x_i, -\frac{g_m(x_i)}{h_m(x_i)}\right\}_{i=1}^n$, al resolver el siguiente problema de optimización

$$\phi_m = \arg\min_{\phi} \sum_{i=1}^{n} \frac{1}{2} h_m(x_i) \left[-\frac{g_m(x_i)}{h_m(x_i)} - \phi(x_i)\right]^2.$$
$$f_m(x) = \alpha \phi_m(x).$$

Luego se actualiza la predicción

$$f_{(m)}(x) = f_{(m-1)}(x) + f_m(x).$$

De esta forma, se obtiene la predicción final

$$f_{(M)}(x) = \sum_{m=0}^{M} f_m(x).$$

Este es el funcionamiento básico del algoritmo, aún así, más adelante se detallará sobre los parámetros extra que incluye la implementación de XGBoost, los cuales sirven para mejorar la forma y modo de aprendizaje a partir de los datos.

2.2. Teoría de Juegos Cooperativos

En esta sección se introducirán los conceptos más importantes de Teoría de Juegos Cooperativos. La importancia de este tema es describir cómo los valores Shapley pueden dar una distribución "justa" de pagos a los jugadores dependiendo de sus contribuciones individuales. Luego, este concepto se llevará al ámbito de ML, donde cada jugador será representado por una variable explicativa.

Definición. Sea $\mathcal{N} = \{1, \ldots, N\}$ un conjunto finito de jugadores. Llamamos a cada subconjunto no vacío $\mathcal{C} \subseteq \mathcal{N}$ una *coalición* de $\mathcal{N}$.

Definición. Un *juego cooperativo* G se define por un par $(\mathcal{N}, v)$, donde $v : 2^{\mathcal{N}} \longrightarrow \mathbb{R}$ es una *función característica* que asigna un número real a cada coalición $\mathcal{C} \subseteq \mathcal{N}$, y que además satisface $v(\emptyset) = 0$. El valor $v(\mathcal{C})$ se conoce como el valor de la coalición $\mathcal{C}$.

Notemos que v asigna el valor de una coalición en su conjunto, y no a sus miembros individuales. Es decir, la función característica no dicta cómo debe dividirse el valor de la coalición $v(\mathcal{C})$ entre los miembros de $\mathcal{C}$. Esta es la principal motivación para abordar el problema de asignación de pagos a partir de cuánto contribuye cada jugador.

Ejemplo. Consideremos un juego cooperativo con 3 jugadores $\mathcal{N} = \{1, 2, 3\}$. La función característica v representa la distribución de pagos para cada coalición, consideremos los

siguientes valores a modo de ejemplo.

$$v(\emptyset) = 0 \qquad v(\{1\}) = 7 \qquad v(\{2\}) = 11 \qquad v(\{3\}) = 14$$
$$v(\{1,2\}) = 18 \qquad v(\{1,3\}) = 21 \qquad v(\{2,3\}) = 23 \qquad v(\{1,2,3\}) = 25.$$

Respecto a los resultados del juego cooperativo, definiremos los siguientes dos conceptos:

Definición. Dado un juego cooperativo $G = (\mathcal{N}, v)$, definimos la *estructura de la coalición* sobre $\mathcal{N}$ como una colección no vacía de coaliciones $C_S = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ tal que

- $\bigcup_{j=1}^{k} \mathcal{C}_j = \mathcal{N}$
- $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset, \ \ \forall i, j \in \{1, \ldots, k\}$ tales que $i \neq j$

Definición. Un vector $\mathbf{z} \in \mathbb{R}^N$ se define como *vector de pago* para la estructura de la coalición C_S sobre $\mathcal{N} = \{1, \ldots, N\}$ si

- $z_i \geq 0, \ \ \forall i \in \mathcal{N}$
- $\sum_{i \in \mathcal{C}_j} z_i \leq v(\mathcal{C}_j), \ \ \forall j \in \{1, \ldots, k\}$

Lo que nos dice esto es que cada jugador debe recibir un pago no negativo y que el monto pagado a una coalición no puede exceder el valor de la coalición, esto ultimo se conoce como condición de *factibilidad*.

El resultado de un juego G es un par $(C_S, \mathbf{z})$, donde C_S es una estructura de coalición sobre G y $\mathbf{z}$ es un vector de pago para C_S. Dado el vector de pago $\mathbf{z}$, $z(\mathcal{C})$ denota el pago total $\sum_{i \in \mathcal{C}} z_i$ a una coalición $\mathcal{C} \subseteq \mathcal{N}$.

Definición. Denotaremos $C_{S_\mathcal{N}}$ al conjunto de estructuras de coalición sobre $\mathcal{N}$.

Ejemplo. Consideremos un juego con 3 jugadores $\mathcal{N} = \{1, 2, 3\}$. Hay 7 posibles coaliciones

no vacías.

$$\begin{aligned}\mathcal{C}_1 &= \{1\}\\ \mathcal{C}_2 &= \{2\}\\ \mathcal{C}_3 &= \{3\}\\ \mathcal{C}_4 &= \{1,2\}\\ \mathcal{C}_5 &= \{2,3\}\\ \mathcal{C}_6 &= \{1,3\}\\ \mathcal{C}_7 &= \{1,2,3\},\end{aligned}$$

y 5 posibles estructuras de coalición

$$C_{S_{\{1,2,3\}}} = \{\ \{\{1\},\{2\},\{3\}\},\ \{\{1\},\{2,3\}\},\ \{\{2\},\{1,3\}\},\ \{\{3\},\{1,2\}\},\ \{\{1,2,3\}\}\ \}.$$

Notemos que toda estructura de coalición junto a un vector de pagos para dicha estructura corresponde a un resultado del juego cooperativo G. Sin embargo, no todos los resultados son igualmente deseables. Por ejemplo, si en un juego todos los jugadores contribuyen igualmente, nos gustaría que la distribución de pagos sea igual para todos y no que un jugador se lleve toda la ganancia.

Aquí es donde introducimos la noción de *valor de Shapley*, el cual es un concepto de solución del juego cooperativo con respecto a la gran coalición $\mathcal{N}$. Este, define una forma de distribuir el valor $v(\mathcal{N})$, basado en la intuición de que el pago que reciba cada jugador debe ser proporcional a su contribución. Antes de definirlo veremos los siguientes conceptos.

Definición. Definimos $\Pi(\mathcal{N})$ como el conjunto de las posibles permutaciones de $\mathcal{N}$. Una permutación específica se define como $\pi \in \Pi(\mathcal{N})$ y $\pi(i)$ es la posición del jugador $i \in \mathcal{N}$ en la permutación π.

Definición. Definimos el *conjunto de predecesores* del jugador $i \in \mathcal{N}$ en la permutación π como la coalición

$$\mathcal{P}_i^\pi := \{j \in \mathcal{N} : \pi(j) < \pi(i)\}$$

Considerando para un juego cooperativo la siguiente permutación $\pi = (3,2,1)$ tendremos los siguientes conjuntos predecesores para $i \in \{1,2,3\}$, $\mathcal{P}_1^\pi = \{3,2\}$, $\mathcal{P}_2^\pi = \{3\}$ y $\mathcal{P}_3^\pi = \emptyset$.

Definición. La *contribución marginal* de un jugador $i \in \mathcal{N}$ con respecto a una permutación π en un juego $G = (\mathcal{N}, v)$ está dada por

$$\Delta_\pi^G(i) := v(\mathcal{P}_i^\pi \cup \{i\}) - v(\mathcal{P}_i^\pi).$$

Esto define en qué medida el jugador i aumenta el valor de la coalición formada por sus predecesores en π cuando se une a ellos. Ahora, definimos el *valor de Shapley* para un jugador i, que básicamente consiste en su contribución marginal promedio, donde el promedio se toma sobre todas las permutaciones de $\mathcal{N}$.

2.2.1. Valor de Shapley

Definición. Dado un juego cooperativo $G = (\mathcal{N}, v)$ con $|\mathcal{N}| = N$, el *valor de Shapley* de un jugador $i \in \mathcal{N}$ denotado por $\varphi_i(G)$ está dado por

$$\varphi_i(G) := \frac{1}{N!} \sum_{\pi \in \Pi(\mathcal{N})} \Delta_\pi^G(i). \tag{2.2.1}$$

Ejemplo. Consideremos un juego $G = (\mathcal{N}, v)$, con 3 jugadores $\mathcal{N} = \{A, B, C\}$, donde la función v tiene los valores $v(\emptyset) = 0$, $v(\{A\}) = v(\{B\}) = v(\{C\}) = 0$, $v(\{A, B\}) = v(\{A, C\}) = 500$, $v(\{B, C\}) = 750$ y $v(\{A, B, C\}) = 1000$. Calculemos el valor Shapley para el jugador A. Vemos que hay $2^{|\mathcal{N}|} = 6$ permutaciones de los jugadores, $\pi_1 = (A, B, C)$, $\pi_2 = (A, C, B)$, $\pi_3 = (B, A, C)$, $\pi_4 = (C, A, B)$, $\pi_5 = (B, C, A)$ y $\pi_6 = (C, B, A)$. Luego, calculamos las contribuciones marginales

$$\begin{aligned}
\Delta_{\pi_1}^G(A) &= v(\{A\}) - v(\emptyset) = 0 \\
\Delta_{\pi_2}^G(A) &= v(\{A\}) - v(\emptyset) = 0 \\
\Delta_{\pi_3}^G(A) &= v(\{A, B\}) - v(\{B\}) = 500 \\
\Delta_{\pi_4}^G(A) &= v(\{A, C\}) - v(\{C\}) = 500 \\
\Delta_{\pi_5}^G(A) &= v(\mathcal{N}) - v(\{B, C\}) = 250 \\
\Delta_{\pi_6}^G(A) &= v(\mathcal{N}) - v(\{B, C\}) = 250
\end{aligned}$$

Finalmente el valor de Shapley para el jugador A será

$$\varphi_A(G) = (500 + 500 + 250 + 250)/6 = 250$$

Ahora, estudiaremos propiedades interesantes y deseables del valor de Shapley. La primera de ellas refiere a que el valor de Shapley es *eficiente*, es decir, distribuye el valor de la gran coalición $v(\mathcal{N})$ a todos los jugadores.

Proposición. Para todo juego cooperativo $G = (\mathcal{N}, v)$ con $\mathcal{N} = \{1, \ldots, N\}$ se cumple

$$\sum_{i=1}^{N} \varphi_i(G) = v(\mathcal{N}).$$

Demostración. Ver Chalkiadakis et al. (2011) (Cap.2, Prop. 2.12).

Otra propiedad interesante del valor de Shapley es que no asigna pago a jugadores que no contribuyen a alguna coalición.

Definición. Dado $G = (\mathcal{N}, v)$ un juego cooperativo, diremos que un jugador i es *dummy* si $v(\mathcal{C}) = v(\mathcal{C} \cup \{i\})$ para toda coalición $\mathcal{C} \subseteq \mathcal{N}$.

Proposición. Para todo juego cooperativo $G = (\mathcal{N}, v)$, si un jugador $i \in \mathcal{N}$ es *dummy* en G, entonces $\varphi_i(G) = 0$.

Demostración. Ver Chalkiadakis et al. (2011) (Cap. 2, Prop. 2.13).

Otra propiedad del valor de Shapley es la simetría. Es decir, que si dos jugadores contribuyen lo mismo, tendrán el mismo valor de Shapley.

Definición. Dado $G = (\mathcal{N}, v)$, decimos que dos jugadores $i, j \in \mathcal{N}$ son *simétricos* en G si $v(\mathcal{C} \cup \{i\}) = v(\mathcal{C} \cup \{j\})$ para cualquier coalición $\mathcal{C} \subseteq \mathcal{N}$.

Proposición. Para todo juego cooperativo $G = (\mathcal{N}, v)$. Si dos jugadores $i, j \in \mathcal{N}$ son simétricos, entonces $\varphi_i(G) = \varphi_j(G)$.

Demostración. Ver Chalkiadakis et al. (2011) (Cap. 2, Prop. 2.14).

2.3. Shapley Additive Explanations (SHAP)

En Lundberg and Lee (2017) se definen los valores SHAP como una medida unificada de la importancia de las variables sobre las predicciones individuales que se obtienen a partir del entrenamiento de un modelo de ML, estos valores representan una solución a la ecuación del valor de Shapley (2.2.1). Dada una instancia del modelo, el objetivo es descomponer la predicción y asignar los valores SHAP a las variables individuales de la instancia. De esta forma, los valores SHAP pueden medir las contribuciones de las variables de entrada a la predicción del modelo (Rozemberczki et al., 2022).

Definición: (*Explicabilidad global*) Sea f el modelo a utilizar y definamos el conjunto de jugadores como los valores de las variables de una única instancia de datos, es decir, $\mathcal{N} = \{x_i \,|\, 1 \leq i \leq n\}$. La función característica v o "pago" a una coalición $\mathcal{C} \subseteq \mathcal{N}$ está dada por la predicción $v(\mathcal{C}) = f(\mathcal{C}) = E[f(x)|x_{\mathcal{C}}]$ calculada desde el subconjunto de variables, donde $E[f(x)|x_{\mathcal{C}}]$ es el valor esperado de la predicción del modelo condicionado por el subconjunto de variables $\mathcal{C}$.

De esta forma, se pueden obtener los valores SHAP extendiendo el cálculo con la ecuación (2.2.1). El cálculo de los valores SHAP ofrece una descomposición completa de la predicción, ya que se satisface la propiedad de eficiencia comentada anteriormente.

A pesar de las ventajas teóricas de utilizar los valores SHAP para la interpretabilidad de un modelo, su uso práctico se ve obstaculizado por los siguientes puntos:

- Es complicado estimar $E[f(x)|x_{\mathcal{C}}]$ eficientemente.
- La ecuación (2.2.1) para el cálculo de los valores SHAP tiene complejidad exponencial.

Si quisiéramos estimar directamente los valores SHAP para un modelo, se puede utilizar Kernel SHAP, un algoritmo descrito en Lundberg et al. (2019). Dicho algoritmo tiene complejidad $\mathcal{O}(TL2^M)$, donde T es el número de árboles del modelo, L el número máximo de hojas en algún árbol y M es el número de variables. Claramente, el algoritmo tiene complejidad exponencial, lo cual es una desventaja si consideramos un número grande de variables. En el mismo trabajo, se propone el método Tree-SHAP, el cual aproxima los valores SHAP mediante un algoritmo de complejidad $\mathcal{O}(TLD^2)$ con $D = \log L$. Es decir, es polinomial con respecto al tamaño de la entrada. Por esto, usualmente se utiliza la implementación de Tree-SHAP para aproximar los valores SHAP. Dicha implementación está incorporada en la librería `shap` (Lundberg and Lee, 2017) de Python. Para más información sobre formas de aproximar los valores SHAP ver Lundberg et al. (2019).

2.4. Métodos de Selección de Variables

El problema de selección de variables surge de la gran cantidad de información existente en las bases de datos actuales, ya que muchos modelos muestran una disminución en la precisión cuando el número de variables utilizadas para entrenarlo es mayor al óptimo (Alsahaf et al., 2022; Kohavi and John, 1997). La presencia de variables irrelevantes implica un aumento en los costes de almacenamiento y computación, además de limitar la interpretabilidad del modelo. La selección de variables puede mitigar estos problemas identificando y seleccionando las variables relevantes y eliminando las irrelevantes y redundantes.

Por esta razón, compararemos entre dos métodos de selección de variables; Boruta (Kursa and Rudnicki, 2010) y Boruta SHAP (Keany, 2020). En el Capítulo 4 se describirá el estudio de simulación realizado con datos generados en distintos escenarios, a modo de validar y comparar la efectividad de ambos métodos. A continuación, definiremos los métodos Boruta y Boruta SHAP.

2.4.1. Boruta

Boruta (Kursa and Rudnicki, 2010) es un método utilizado para obtener un subconjunto de variables "importantes" para el modelo. Está basado en el algoritmo Random Forest (Breiman, 2001), el cual es relativamente rápido y normalmente puede ejecutarse sin necesidad de ajustar los hiperparámetros. Además, proporciona una estimación numérica de la importancia de las variables que se calcula por separado para todos los árboles del bosque que utilizan una variable determinada. Por otro lado, teniendo la importancia entregada por Random Forest se calculan los puntajes estándar

$$Z_i = \frac{x_i - \mu}{\sigma},$$

Para $i \in V$, con V el conjunto de variables del modelo, μ la media y σ desviación estándar de las importancias de las variables. Estos puntajes estándar miden qué tan lejos está la importancia de la i-ésima variable con respecto a la desviación estándar de todas las importancias. Este puntaje será utilizado por Boruta como medida de importancia para discernir entre las variables importantes y no importantes. En términos generales este método se basa en la intuición de que una variable es útil sólo si es capaz de hacerlo mejor que la mejor variable aleatorizada.

El algoritmo Boruta consiste en los siguientes pasos:

(1) Se extiende la tabla de datos con copias de todas las variables explicativas, es decir, si la

tabla original tiene dimensión $n \times m$ (con n observaciones y m variables), el sistema extendido tendrá dimensión $n \times 2m$. Llamaremos variables *Shadow* a las agregadas al sistema.

(2) A las variables *Shadow* se les somete a una aleatorización por columnas, con el fin de eliminar la correlación con la variable de respuesta.

(3) Se ejecuta una instancia de Random Forest en los datos extendidos y se calculan los puntajes Z (medidas de importancia para las variables explicativas).

(4) Se encuentra el máximo puntaje Z entre todas las variables *Shadow* (que definiremos como $\gamma := \max_{i \in S} Z_i$), y luego se agrega un conteo a cada variable i que cumpla con la condición $Z_i > \gamma$.

(5) Para cada variable explicativa con importancia desconocida, se realiza una prueba de igualdad de dos colas con respecto a γ.

(6) Considerar como "no importantes" a las variables que tienen una importancia significativamente inferior al γ y eliminarlos permanentemente de la tabla.

(7) Considerar "importantes" a las variables que tienen una importancia significativamente mayor a γ.

(8) Eliminar todas las variables *Shadow*.

(9) Repetir el procedimiento hasta que se asigne la importancia a todas las variables.

Cabe destacar que el algoritmo Boruta está construido para trabajar sobre un modelo Random Forest, sin embargo, esto no limita su uso a distintos algoritmos que proporcionen una importancia de las variables. Para efectos de esta tesis se utilizará el modelo XGBoost en su reemplazo, ya que Random Forest no admite datos perdidos o NaN. Se modificará el código de la librería BorutaPy para hacerla compatible con XGBoost (Alsahaf et al., 2022). El procedimiento sigue siendo el mismo en cuanto al algoritmo. De aquí en adelante nos referiremos a Boruta como Boruta-XGBoost.

2.4.2. BorutaShap

BorutaShap (Keany, 2020) es un método de selección de variables basado en Boruta, utiliza el mismo algoritmo descrito en la Sección 2.4.1, con la salvedad de que utiliza SHAP como método de importancia de variables en reemplazo de la importancia entregada por el algoritmo Random Forest. Esto permite utilizar cualquier algoritmo basado en árboles que sea compatible con SHAP. Así, utilizaremos BorutaShap mediante XGBoost.

Según Keany (2020), BorutaShap tiene una mayor precisión en la selección de variables comparado con Boruta, esto, considerando Boruta mediante Random Forest. En este trabajo de tesis haremos una comparación entre Boruta mediante XGBoost (con la importancia de variables entregada por XGBoost) y BorutaShap (con la importancia de variables entregada por SHAP), ya que no se encontró dicha comparación al momento de buscar literatura al respecto.

2.5. Búsqueda de Hiperparámetros

Los hiperparámetros juegan un rol muy importante en el entrenamiento de cualquier algoritmo o modelo de ML (Salas et al., 2022), ya que estos controlan directamente el aprendizaje de los modelos y pueden tener un gran efecto en el rendimiento de estos (Wu et al., 2019). Se define el problema de búsqueda de hiperparámetros como encontrar una selección óptima de parámetros de forma que al entrenar un modelo se trate de minimizar una función de error. Los métodos más comunes para la selección de hiperparámetros son la búsqueda por cuadrícula (o grid search), en donde se elige manualmente un conjunto de parámetros sobre los cuales se itera y se llega a los óptimos mediante la minimización del error. La búsqueda aleatoria (random search), que sigue la misma lógica, pero en vez de elegir la serie de hiperparámetros, estos son elegidos de manera aleatoria. Otro método de búsqueda es el de Optimización Bayesiana, el cual consiste en crear un modelo probabilístico en donde la función objetivo es la métrica de evaluación del modelo. Así, se consigue que la búsqueda se vaya dirigiendo en cada iteración hacia las regiones de mayor interés. Según Bergstra and Bengio (2012), las estrategias de búsqueda en cuadrícula son inferiores a la búsqueda aleatoria, y sugiere el uso de la Optimización Bayesiana, la cual se utilizará en esta tesis mediante la librería `HyperOpt` de Python.

2.5.1. Optimización Bayesiana

La Optimización Bayesiana es un método eficaz para encontrar los extremos de alguna función computacionalmente costosa. El objetivo de la optimización es encontrar el máximo para una función desconocida f:

$$x^+ = \arg\max_{x \in A} f(x)$$

donde A es el espacio de búsqueda de x. La optimización Bayesiana surge a partir del Teorema de Bayes, esto es:

$$\mathbb{P}(Score|Hiperparmetros) = \frac{\mathbb{P}(Hiperparmetros|Score) * \mathbb{P}(Score)}{\mathbb{P}(Hiperparmetros)},$$

es decir, se combina la distribución a priori de la función con su información muestral (evidencia) para obtener así información posterior de la función. A continuación, la información posterior se utiliza para encontrar el punto en el que la función se maximiza (o minimiza) según algún criterio.

2.6. Métricas de evaluación

Además de las métricas comunes de evaluación de modelos de regresión, como lo son R^2, error cuadrático medio (RMSE) y error absoluto medio (MAE), se definen las siguientes métricas utilizadas por la institución financiera, las cuales dan un intervalo de confianza en que puede caer la variación porcentual entre la predicción y el valor real.

2.6.1. Porcentajes de éxito, subestimación y sobrestimación

El *porcentaje de éxito* se refiere al porcentaje de casos tal que la variación entre la predicción del modelo y el valor real de la variable objetivo queda dentro del intervalo $[-x, x]$.

Con la misma lógica se define el *porcentaje de subestimación* (*sobrestimación*), como el porcentaje de casos tal que la variación entre la predicción del modelo y el valor real queda sobre (bajo) el intervalo $[-x, x]$.

Más formalmente, se define el conjunto P como las diferencias porcentuales entre el valor predicho $\hat{y}_i$ del modelo con el valor real y_i de la variable objetivo, para cada observación en el conjunto que se evalúe.

$$P := \left\{ p_i \mid p_i = \frac{\hat{y}_i - y_i}{y_i} \times 100,\ i = 1, \ldots, N \right\}$$

Luego, definimos los siguientes conjuntos con los casos de éxito E, casos de subestimación S^- y casos de sobrestimación S^+.

$$E = \{p_i \in P : -x \leq p_i \leq x\}$$

$$S^- = \{p_i \in P : p_i < -x\}$$

$$S^+ = \{p_i \in P : p_i > x\}$$

Finalmente, los porcentajes de éxito, subestimación y sobrestimación vienen dados por

$$\%E = \frac{|E|}{|E| + |S^-| + |S^+|} \times 100$$
$$\%S^- = \frac{|S^-|}{|E| + |S^-| + |S^+|} \times 100$$
$$\%S^+ = \frac{|S^+|}{|E| + |S^-| + |S^+|} \times 100$$

Capítulo 3

Metodología

En este capítulo se describe la metodología utilizada para el entrenamiento de los modelos XGBoost.

En primer lugar, se realizó el entrenamiento de modelos considerando las 426 variables explicativas originales del problema. Luego, una vez realizada la selección de variables por medio de Boruta-XGBoost y BorutaShap se procedió a entrenar modelos que consideraran esta reducción en la dimensionalidad del problema. En cada uno de los modelos se dividió la base de datos, considerando un 80 % de las observaciones para entrenamiento y el restante 20 % como conjunto de testeo, a modo de medir el poder predictivo en un conjunto independiente (no observado por el modelo) del utilizado para el entrenamiento. Los hiperparámetros utilizados en cada modelo fueron obtenidos con `HyperOpt` y se pueden ver en la Tabla 3.0.1. Finalmente, a modo de resumen, en 3.0.1 se muestra un diagrama con la metodología propuesta en este trabajo de tesis.

A continuación se detallan los modelos entrenados.

- Modelo 1: El primer modelo considera las 426 variables originales de la base y se utilizan los hiperparámetros por defecto de la librería `xgboost` de Python.
- Modelo 2: El segundo modelo considera las 426 variables originales de la base y los hiperparámetros obtenidos a través de 10 iteraciones de `HyperOpt`.
- Modelo 3: El tercer modelo considera 35 variables obtenidas a través del método BorutaShap de selección de variables. Los hiperparámetros se obtienen a través de 10 iteraciones de `HyperOpt`.
- Modelo 4: El cuarto modelo considera 28 variables obtenidas a través del método Boruta-XGBoost de selección de variables. Los hiperparámetros se obtienen a través de 10 iteraciones de `HyperOpt`.

A continuación, se definen los hiperparámetros de un modelo XGBoost.

- `objective`: Parámetro que especifica el tipo de aprendizaje del modelo, puede variar entre distintos tipos de regresión para variables de respuesta continua, tanto como para distintos tipos de clasificación para variables de respuesta binarias o multiclase.
- `alpha`, `lambda`: Parámetros de regularización que penalizan la complejidad del modelo.
- `colsample_bytree`: Parámetro entre 0 y 1 que representa la proporción de columnas muestreadas para construir cada árbol.
- `gamma`: Parámetro que representa el valor mínimo posible de la función de pérdida para realizar una nueva partición en un nodo del árbol.
- `learning_rate`: Parámetro utilizado para la reducción de los pesos de las variables en cada iteración del algoritmo, sirve para evitar el sobreajuste.
- `max_depth`: Parámetro que indica la máxima profundidad de los árboles dentro del modelo. Si se considera una alta profundidad, el modelo tenderá a sobreajustarse y será más costoso de computar.
- `min_child_weight`: Parámetro que indica la suma mínima de los pesos para seguir particionando el árbol.
- `n_estimators`: Número de árboles dentro del ensamble.
- `subsample`: Parámetro que indica proporción de muestreo del conjunto de entrenamiento.

Hiperparámetros	Modelo 1	Modelo 2	Modelo 3	Modelo 4
`objetive`	reg:squarederror	reg:squarederror	reg:squarederror	reg:squarederror
`alpha`	0	0.00058	17.68	0.091
`colsample_bytree`	1	0.648	0.515	0.580
`gamma`	0	4635	0.061	0.00038
`lambda`	1	2.733	7.429	1.016
`learning_rate`	0.3	0.134	0.0429	0.0094
`max_depth`	6	5	82	40
`min_child_weight`	1	2.988	19.09	7.195
`n_estimators`	100	580	397	949
`subsample`	1	0.796	0.619	0.776

Tabla 3.0.1: Hiperparámetros para entrenamiento de modelos XGBoost.

El entrenamiento de los modelos XGBoost se llevaron a cabo en una máquina virtual de Google Cloud Platform de 16 vCPUs y 60 GB de memoria RAM.

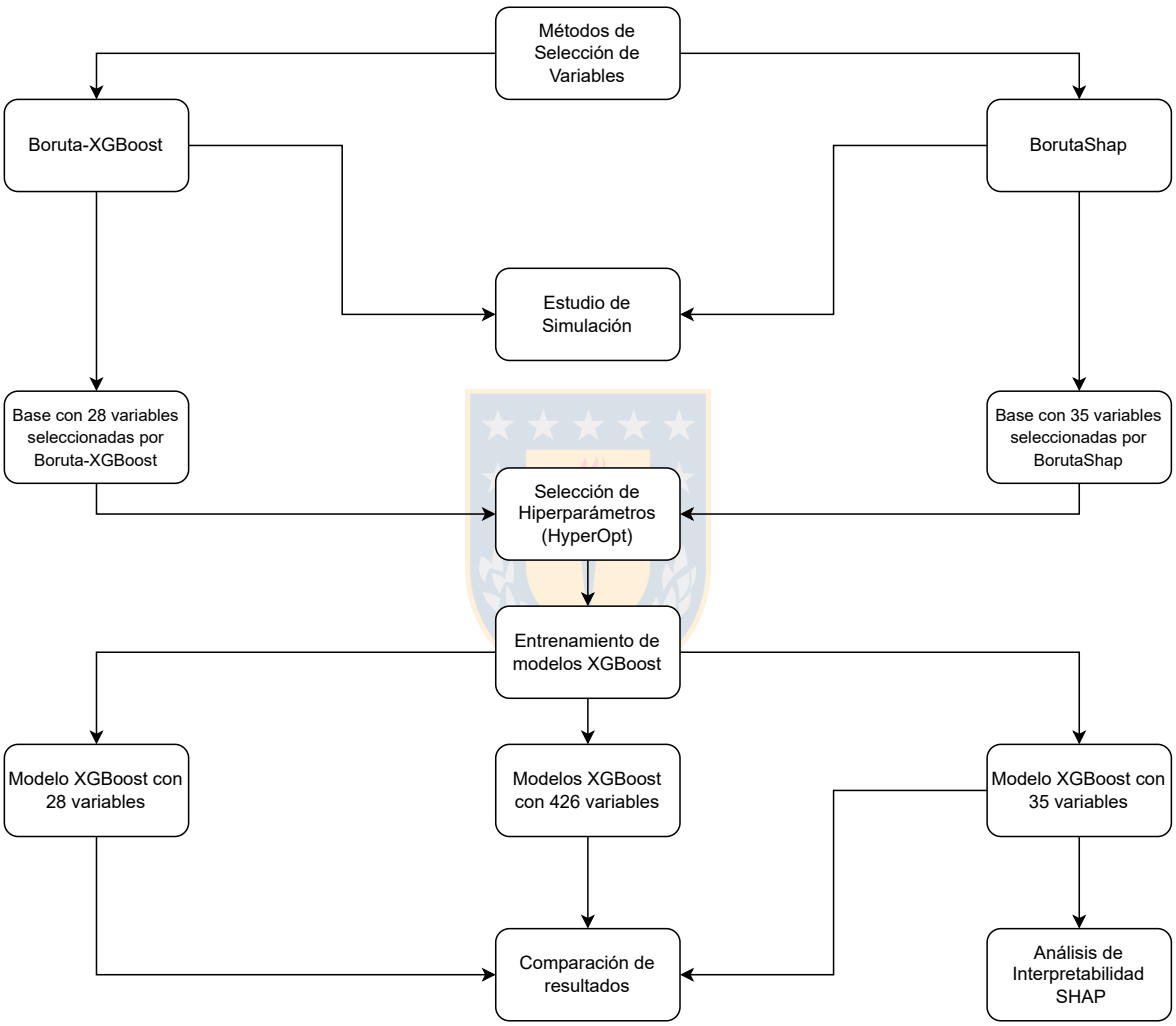


Figura 3.0.1: Diagrama con metodología propuesta.

Capítulo 4

Estudio de Simulación

En este capitulo se describe el estudio de simulación realizado, el cual tuvo por objetivo validar y comparar la capacidad de reconocer la importancia de las variables explicativas de dos métodos de selección de variables (Boruta-XGBoost (BXG) y BorutaShap (BSHAP)) en un ambiente controlado.

Se definieron tres escenarios, considerando distintas estructuras de datos artificiales. Las observaciones fueron generadas siguiendo la siguiente secuencia; Se fija un número n de observaciones a generar, se fijan $p, q \in \mathbb{N}$, donde $p+q$ es el número de variables explicativas. Las variables $x_1, \ldots, x_p, x_{p+1}, \ldots, x_{p+q}$ son generadas aleatoriamente, donde la mitad de ellas se distribuyen $N(0,1)$, mientras que la otra mitad se distribuyen Beta$(1,6)$. Las distribuciones y parámetros propuestos en cada caso se definieron luego de analizar el comportamiento de las variables originales en la base de datos real. Por otra parte, la ventaja de nuestro estudio de simulación es la inclusión de valores NaN dentro de las distintas variables, mediante lo anterior buscamos construir conjuntos de datos sintéticos que sean lo más parecidos a los datos reales que disponen las entidades bancarias. Finalmente, las p primeras variables están directamente relacionadas a la variable objetivo Y de la forma

$$Y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p,$$

donde $\alpha, \beta_1, \ldots, \beta_p$ son parámetros fijos. A continuación, se muestra el pseudocódigo de la función generadora de datos pseudo aleatorios para cada escenario.

Algorithm GENERAR_DATOS(α, β, n, p, q, NaN=False)

1: Input: $\alpha \in \mathbb{R}$, $n, p, q \in \mathbb{N}$, $\beta \in \mathbb{R}^{1\times p}$
2: **for** i, j **in** $(1, \ldots, p), (p+1, \ldots, q)$ **do**
3: $x_i \leftarrow$ RandomNorm$(0, 1, size = n)$
4: $x_j \leftarrow$ RandomNorm$(0, 1, size = n)$
5: **end for**
6: $rand\_ind_1 \leftarrow$ SAMPLE$([1, \ldots, p], size = 0{,}5p)$
7: $rand\_ind_2 \leftarrow$ SAMPLE$([p+1, \ldots, q], size = 0{,}5q)$
8: **for** i, j **in** $rand\_ind_1, rand\_ind_2$ **do**
9: $x_i \leftarrow$ RandomBeta$(1, 6, size = n)$
10: $x_j \leftarrow$ RandomBeta$(1, 6, size = n)$
11: **end for**
12: $y = \alpha + (x_1, \ldots, x_p) \cdot \beta$
13: $X = [\{x_1, \ldots, x_p\}, \{x_{p+1}, \ldots, x_{p+q}\}]$
14: **if** NaN=True **then**
15: **for** col **in** X.columns **do**
16: col.SAMPLE(frac=0.3) = NaN
17: **end for**
18: **end if**
19: **return** $[y, X]$

El algoritmo GENERAR_DATOS funciona de la siguiente manera; En las líneas 2, 3 y 4 se generan $p + q$ columnas de datos aleatorios que distribuyen Normal(0,1). Se hace la distinción entre las columnas que van de 1 a p y las que van de $p + 1$ a q. En las lineas 6 y 7 se definen muestras aleatorias para los índices i y j, cada uno de estos de tamaño $p/2$ y $q/2$ respectivamente. En las lineas 8, 9 y 10 se generan datos aleatorios que distribuyen Beta(1,6) para los índices aleatorios definidos anteriormente. La idea es que la mitad de las variables distribuyan Normal(0,1) y la otra mitad distribuya Beta(1,6), ambas con índices aleatorios. En la linea 12 se forma la variable de respuesta y que depende únicamente de las variables $x_1, \ldots, x_p$. En la línea 13 se concatenan las columnas que dependen de y con las independientes de y. De la linea 14 a la 17 corresponde al caso en que se agreguen datos NaN, en dicho caso, se le asigna NaN a una muestra aleatoria de los datos de tamaño $30\,\%$ de la base total.

Ejemplo: Considerando $p = 3$, $q = 2$, $n = 5$, $\alpha = 0{,}8$, $\beta_1 = 0{,}5$, $\beta_2 = 1$ y $\beta_3 = 1{,}1$ obtenemos la siguiente tabla de datos aleatorios. Ver Tabla 4.0.1.

La idea principal de este estudio de simulación es ver si los métodos son capaces de discernir entre elegir como importantes las variables $x_1, \ldots, x_p$ que influyen explícitamente sobre la variable de respuesta Y, y considerar como no importantes las variables $x_{p+1}, \ldots, x_{p+q}$.

Y	x1	x2	x3	x4	x5
4.33	0.02	2.01	-0.08	0.47	0.34
2.68	0.10	0.37	-0.12	0.90	0.05
0.46	0.05	-0.52	-1.31	0.96	0.12
2.49	0.14	1.35	-1.20	0.31	0.01
1.26	0.57	0.29	-1.56	-0.71	0.16

Tabla 4.0.1: Ejemplo de datos generados.

La simulación contempló tres escenarios, los cuales se describen a continuación:

- Escenario 1: $p = 5$, $q = 5$, $n = 100$, $\alpha = 1$, $\vec{\beta} = \{0{,}5; 1; 1{,}5; 2; 2{,}5\}$
- Escenario 2: $p = 3$, $q = 9$, $n = 500$, $\alpha = 1$, $\vec{\beta} = \{0{,}2; 0{,}6; 0{,}9\}$
- Escenario 3: $p = 6$, $q = 18$, $n = 500$, $\alpha = 1$, $\vec{\beta} = \{0{,}5; 0{,}75; 1; 1{,}25; 1{,}50; 1{,}75\}$

Para cada Escenario se realizaron 100 iteraciones de BXG y BSHAP, donde en cada iteración se generaron nuevas observaciones. Luego, se realizó un conteo de veces en que cada algoritmo consideraba importante cada variables explicativa y además se calculó la cobertura de cada variable, la cual definimos como el porcentaje de veces en que cada variable fue seleccionada dentro del conteo total. Además, cabe destacar que para cada escenario se consideró casos con datos conteniendo un 30 % de observaciones NaN, considerando que los datos reales poseen datos del tipo NaN. A continuación, se muestra el pseudocódigo de las simulaciones para Boruta-XGBoost y BorutaShap.

Algorithm SIMULACIÓN_BORUTA$(M, \alpha, \beta, n, p, q)$

Input: M número de iteraciones, $\alpha \in \mathbb{R}$, $n, p, q \in \mathbb{N}$, $\beta \in \mathbb{N}^{1\times p}$
$var\_imp = [\,]$, $var\_imp\_nan = [\,]$
for $i = 1, \ldots, M$ **do**
    $var\_imp \leftarrow$ BORUTA-XGBOOST(GENERAR_DATOS(α, β, n, p, q))
    $i \leftarrow i + 1$
end for
$var\_imp \leftarrow$ GROUP(var_imp)
for $i = 1, \ldots, M$ **do**
    $var\_imp\_nan \leftarrow$ BORUTA-XGBOOST(GENERAR_DATOS$(\alpha, \beta, n, p, q,\ NaN$=True))
    $i \leftarrow i + 1$
end for
$var\_imp\_nan \leftarrow$ GROUP(var_imp_nan)
return var_imp, var_imp_nan

El algoritmo SIMULACIÓN_BORUTA funciona de la siguiente manera; Primero, se define M

como el número de iteraciones de la simulación y las variables α, β, n, p, q. Entre las lineas 3 y 5 se ejecuta M veces la función BORUTA-XGBOOST, que retorna un conjunto de variables importantes dado una tabla de datos. En este caso, la función GENERAR_DATOS entrega datos aleatorios en cada iteración de la simulación. En la linea 7 se agrupan todas las variables importantes resultantes de las M iteraciones para su conteo. Desde las lineas 8 a 12 se repite el mismo procedimiento, con la salvedad de que la función GENERAR_DATOS retorna datos incluyendo $30\,\%$ de datos NaN. Finalmente, en la linea 13 se retornan los vectores var_imp y var_imp_nan, que corresponden a la cantidad de veces que cada variable fue seleccionada como importante por Boruta-XGBoost.

Para la simulación con BorutaShap el procedimiento es el mismo que en el algoritmo SIMULACIÓN_BORUTA, basta reemplazar BORUTA-XGBOOST por BORUTASHAP en las líneas 4 y 9 del algoritmo.

Para la aplicación de los métodos de selección de variables se utilizaron las librerías `Boruta` y `BorutaShap` en Python, las cuales fueron modificadas para permitir datos con valores NaN y para ser compatibles con el modelo XGBoost. Como mencionamos anteriormente, ambos métodos los basamos en XGBoost, por lo que para cada iteración de la simulación se ejecuta una instancia de `HyperOpt` para la selección óptima de hiperparámetros del modelo.

Las simulaciones se realizaron en un MacBook Pro M1 2020 de 8 GB de memoria RAM.

4.1. Resultados

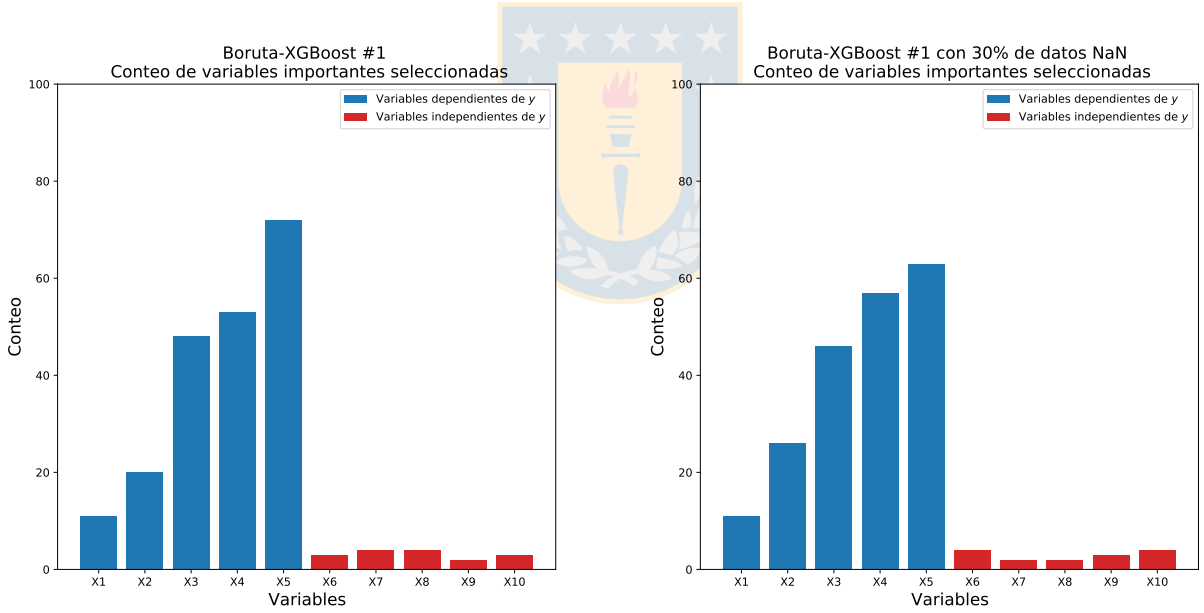
Luego de ejecutar 100 veces Boruta-XGBoost (BXG) y BorutaShap (BSHAP) sobre los datos controlados, obtuvimos los siguientes resultados:

- Para el Escenario 1 vemos que BSHAP es capaz de seleccionar más veces como importantes las variables $x_1, \ldots, x_5$, en comparación de BXG, tanto para el caso sin datos NaN y con 30 % de datos NaN, como se observa en 4.1.1 y 4.1.2. Dentro de BSHAP, vemos que este selecciona de manera mas uniforme las variables importantes en el caso de datos NaN. Además, cabe descatar que selecciona en mayor cantidad variables no importantes en comparación a BXG.
- Para el Escenario 2 vemos que BSHAP selecciona más veces como importantes las variables x_1, x_2 y x_3 en comparación a BXG, para los casos con y sin datos NaN (4.1.3, 4.1.4). No obstante y tal como en el Escenario 1, BSHAP selecciona una mayor cantidad de veces aquellas variables no importantes, en comparación con BXG. Esto se puede

observar también en la Tabla 4.1.3.

- En el Escenario 3 vemos que se repite el mismo comportamiento, haciéndose más notoria la diferencia a favor de BSHAP, que es capaz de seleccionar más veces como importantes las variables $x_1, \ldots, x_6$, en comparación a BXG. Por otro lado, respecto a la tabla de coberturas para el Escenario 3 4.1.4 vemos valores similares para las variables importantes en el caso con datos NaN, incluso una mayor cobertura de BXG para x_3 y x_4, lo cual a priori no se condice con los gráficos de conteos 4.1.5 y 4.1.6. Esto se puede explicar debido a que la cobertura se calcula en base a las veces en que los métodos seleccionaron variables importantes dentro de todas las variables, pudiendo en ciertas iteraciones no seleccionar ninguna.

Finalmente, al analizar los tiempos de ejecución de cada método, vemos a partir de los datos presentados en la Tabla 4.1.1 que BSHAP tarda un tiempo mucho mayor cuando aumenta el número de variables, con un tiempo de 11.921 segundos (3.3 horas) en comparación de un tiempo de 959 segundos (0.26 horas) para BXG en el Escenario 3. De forma gráfica se pueden ver los tiempos de ejecución en la Figura 4.1.7.

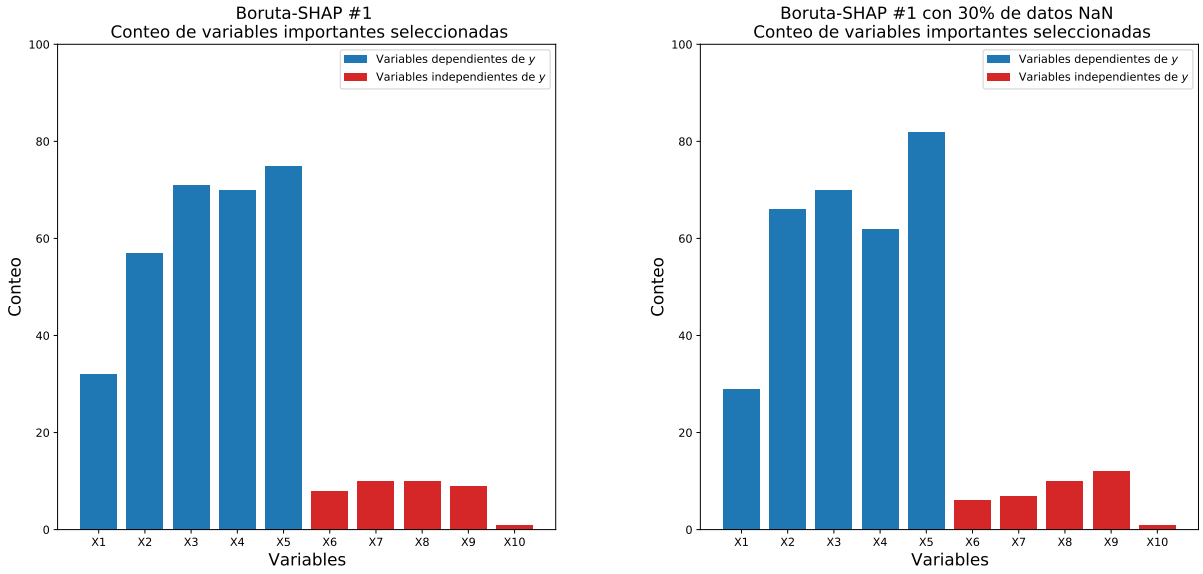


(a) (b)

Figura 4.1.1: Resultados de simulación #1 con BXG.

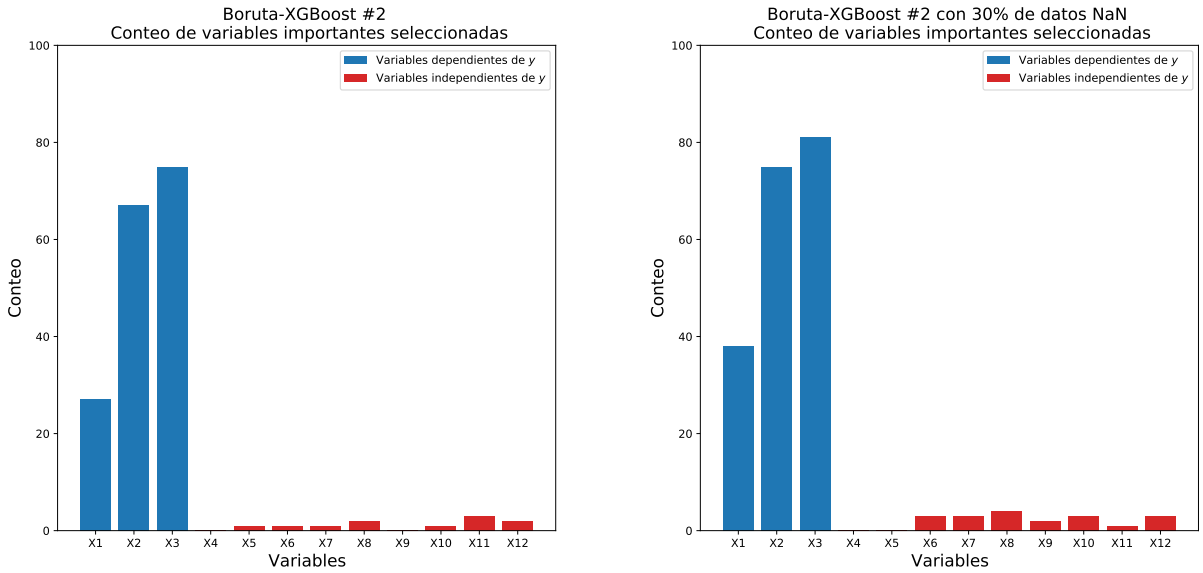
	Escenario 1	Escenario 2	Escenario 3
BXG	132s	524s	959s
BSHAP	1833s	6737s	11921s

Tabla 4.1.1: Tiempos de ejecución de simulaciones en segundos.



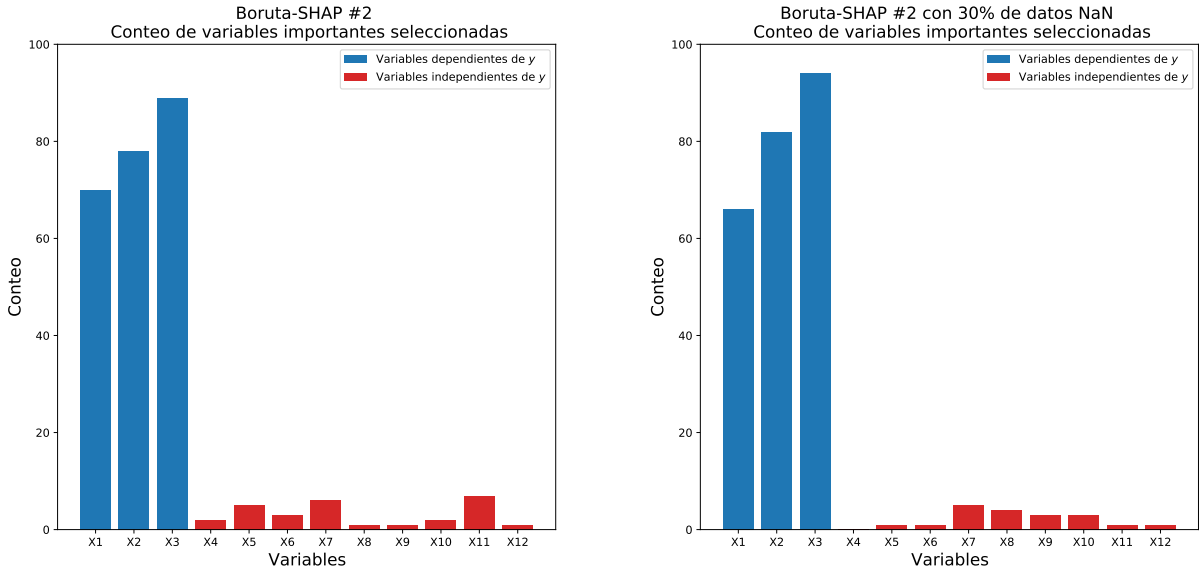
(a) (b)

Figura 4.1.2: Resultados de simulación #1 con BSHAP.



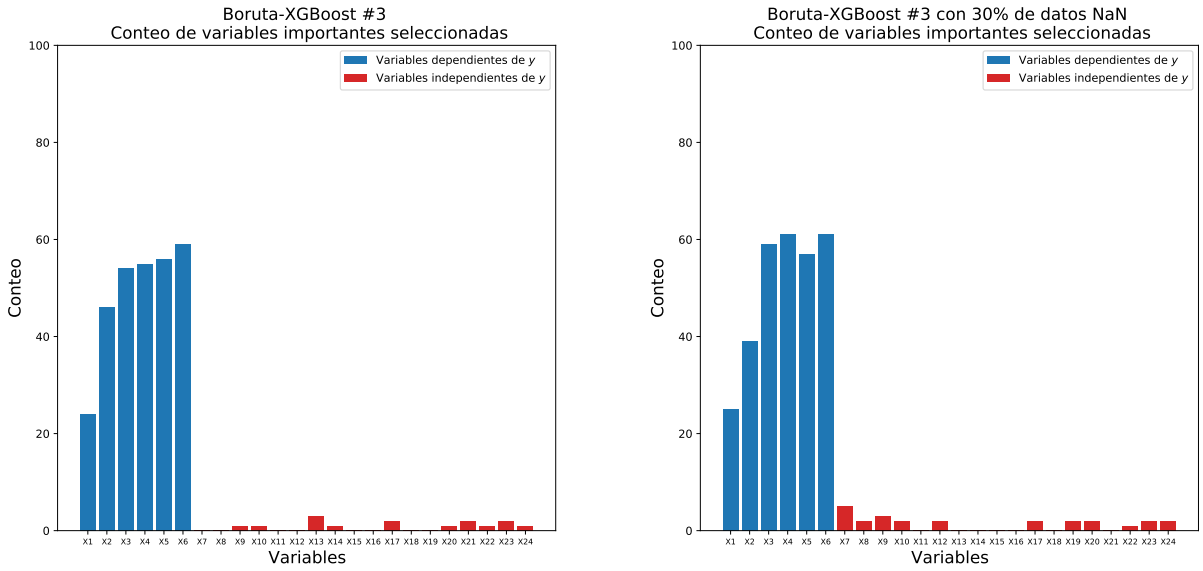
(a) (b)

Figura 4.1.3: Resultados de simulación #2 con BXG.



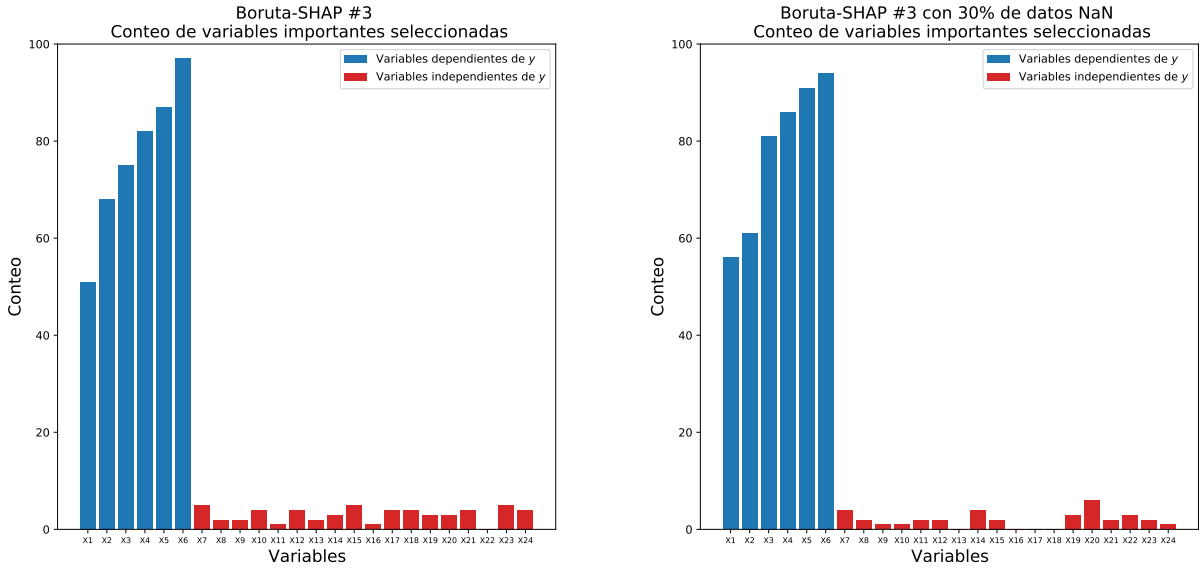
(a) (b)

Figura 4.1.4: Resultados de simulación #2 con BSHAP.



(a) (b)

Figura 4.1.5: Resultados de simulación #3 con BXG.



(a) (b)

Figura 4.1.6: Resultados de simulación #3 con BSHAP.

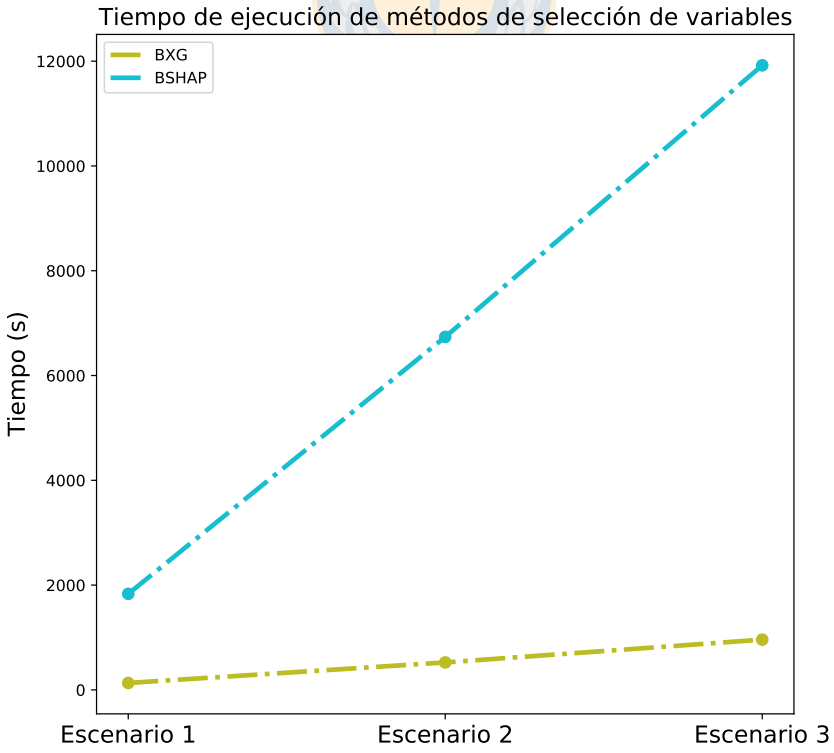


Figura 4.1.7: Tiempos de ejecución de BXG y BSHAP en distintos escenarios.

	BXG		BSHAP	
	0 % NaN	30 % NaN	0 % NaN	30 % NaN
x1	5.00	5.05	9.33	8.41
x2	9.09	11.93	16.62	19.13
x3	21.82	21.10	20.70	20.29
x4	24.09	26.15	20.41	17.97
x5	32.73	28.90	21.87	23.77
x6	1.36	1.83	2.33	1.74
x7	1.82	0.92	2.92	2.03
x8	1.82	0.92	2.92	2.90
x9	0.91	1.38	2.62	3.48
x10	1.36	1.83	0.29	0.29

Tabla 4.1.2: Cobertura (%) de variables para escenario 1.

	BXG		BSHAP	
	0 % NaN	30 % NaN	0 % NaN	30 % NaN
x1	15.00	17.84	26.42	25.29
x2	37.22	35.21	29.43	31.42
x3	41.67	38.03	33.58	36.02
x4	0.00	0.00	0.75	0.00
x5	0.56	0.00	1.89	0.38
x6	0.56	1.41	1.13	0.38
x7	0.56	1.41	2.26	1.92
x8	1.11	1.88	0.38	1.53
x9	0.00	0.94	0.38	1.15
x10	0.56	1.41	0.75	1.15
x11	1.67	0.47	2.64	0.38
x12	1.11	1.41	0.38	0.38

Tabla 4.1.3: Cobertura (%) de variables para escenario 2.

	BXG		BSHAP	
	0 % NaN	30 % NaN	0 % NaN	30 % NaN
x1	7.77	7.65	9.88	11.11
x2	14.89	11.93	13.18	12.10
x3	17.48	18.04	14.53	16.07
x4	17.80	18.65	15.89	17.06
x5	18.12	17.43	16.86	18.06
x6	19.09	18.65	18.80	18.65
x7	0.00	1.53	0.97	0.79
x8	0.00	0.61	0.39	0.40
x9	0.32	0.92	0.39	0.20
x10	0.32	0.61	0.78	0.20
x11	0.00	0.00	0.19	0.40
x12	0.00	0.61	0.78	0.40
x13	0.97	0.00	0.39	0.00
x14	0.32	0.00	0.58	0.79
x15	0.00	0.00	0.97	0.40
x16	0.00	0.00	0.19	0.00
x17	0.65	0.61	0.78	0.00
x18	0.00	0.00	0.78	0.00
x19	0.00	0.61	0.58	0.60
x20	0.32	0.61	0.58	1.19
x21	0.65	0.00	0.78	0.40
x22	0.32	0.31	0.00	0.60
x23	0.65	0.61	0.97	0.40
x24	0.32	0.61	0.78	0.20

Tabla 4.1.4: Cobertura (%) de variables para escenario 3.

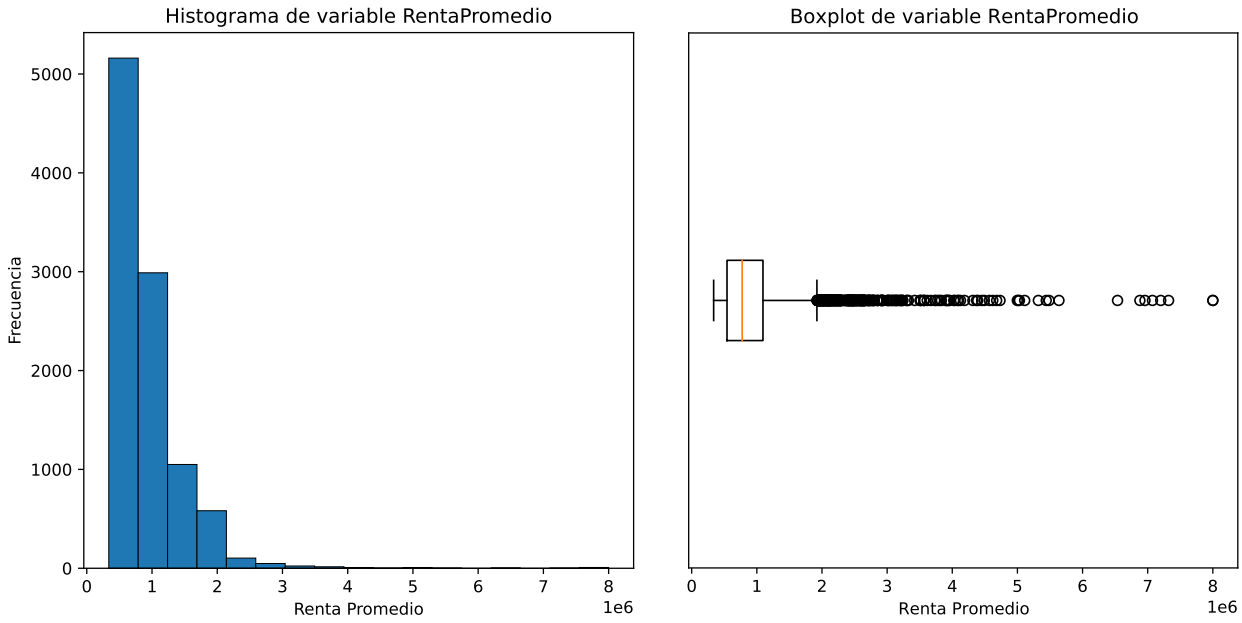
Capítulo 5

Aplicación

En este capítulo se muestran los resultados de una aplicación empírica, la cual se compone de un set de datos facilitado por una entidad bancaria chilena anónima. Primero que todo se presenta una breve descripción de los datos y luego se muestran los resultados asociados a los objetivos trazados en este trabajo (entrenamiento de modelos XGBoost, selección de variables, interpretabilidad de predicciones usando el método SHAP).

5.1. Descripción de la base de datos

La base de datos a utilizar cuenta con 10.000 observaciones (registros de clientes), 426 variables explicativas y una variable de respuesta llamada *Renta Promedio*, la cual corresponde al promedio de renta de 3 meses consecutivos de cada cliente. Por temas de confidencialidad de los datos, cambiamos el nombre de las variables explicativas por $x_1, x_2, \ldots, x_{426}$, en reemplazo de sus nombres originales. En la Tabla 5.1.1 se muestran algunas características de la variable de respuesta *Renta Promedio*, además de su histograma de frecuencias y Boxplot en la Figura 5.1.1. De la Figura 5.1.1a notamos que la variable objetivo tiene una distribución asimétrica con su mayoría de observaciones concentradas en montos bajos de renta, para luego decaer rápidamente hasta las observaciones de mayor renta promedio. Las 426 variables explicativas provienen de diversas fuentes de datos internas de la institución financiera. Para hacer más preciso el posterior análisis de interpretabilidad, definimos $F_1, F_2, \ldots, F_{14}$ como las distintas fuentes de información internas de la institución financiera de donde provienen las variables. En la Figura 5.1.2 se muestra como distribuyen las 426 variables en las 14 fuentes de información. Respecto de las variables explicativas, un 31.70 % de los datos corresponde a un valor NaN. Estos valores perdidos son relevantes para la institución financiera, por lo que no serán descartados del análisis ni del entrenamiento de los modelos. En lo que sigue, se mostrarán los resultados de aplicar los métodos Boruta-XGBoost y BorutaShap sobre los datos, además de los resultados de los modelos XGBoost entrenados. Finalmente, se hará el



(**a**) Distribución de variable objetivo. (**b**) Boxplot de variable objetivo.

Figura 5.1.1: Características de variable RentaPromedio

análisis de interpretabilidad al modelo que logre el mejor poder predictivo.

Promedio	$903.330
Desviación estándar	$552.549
P25	$539.997
P50	$772.076
P75	$1.092.052

Tabla 5.1.1: Características de la variable Renta Promedio

5.2. Resultados

5.2.1. Selección de variables y entrenamiento de XGBoost

Al aplicar BorutaShap sobre los datos reales se obtuvieron 35 variables seleccionadas de las 426 originales, las cuales se distribuyen de la siguiente manera en las distintas fuentes de información (Figura 5.2.1). Notamos que la mayoría de variables seleccionadas como importantes provienen de las fuentes F_5, F_8 y F_9, en contraste con las variables originales, que provienen en mayor número desde las fuentes F_1, F_9 y F_{13}. Es decir, una mayor cantidad de variables en una misma fuente de información no implica que estas aporten más a la predicción del modelo. Por otro lado, los grupos de variables F_5, F_8 y F_9 corresponden a información del sistema financiero e información de saldos y movimientos transaccionales de clientes del banco.

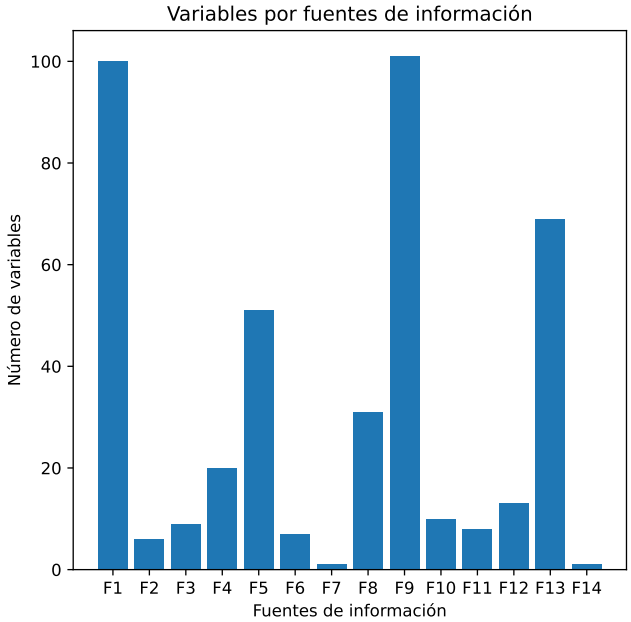


Figura 5.1.2: Gráfico de variables por fuentes de información.

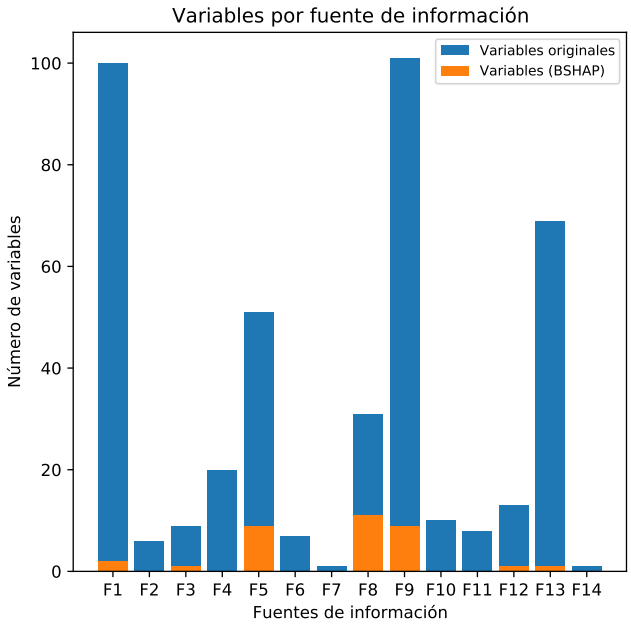


Figura 5.2.1: Gráfico de variables seleccionadas por BorutaShap por fuentes de información.

Respecto a los modelos XGBoost, notamos en primer lugar que se obtienen mejores resultados de predicción al utilizar los hiperparámetros encontrados mediante HyperOpt en comparación a los hiperparámetros por defecto (Tablas 5.2.1, 5.2.2), pudiendo aumentar el porcentaje de éxito desde 33.3 % a 34.15 %, además de disminuir el porcentaje de subestimación de 27.8 % a un 26.85 %. Por otra parte, de las Tablas 5.2.2 y 5.2.3 notamos diferencias entre el modelo XGBoost entrenado con las 426 variables (Modelo 2) en comparación con el modelo XGBoost entrenado con las 35 variables obtenidas por BorutaShap (Modelo 3). Primero con $\pm 15\,\%$ de confianza vemos un aumento del 34.15 % al 35.15 % de porcentaje de éxito. También, una disminución en el porcentaje de subestimación, de 26.85 % a 26.75 %, y de un 39.0 % a un

38.1 % de sobrestimación. Luego, si vemos el caso de $\pm 25\,\%$ de confianza vemos un aumento en el éxito de 53.35 % a 54.35 %, un aumento de la subestimación de 16.95 % a un 17.1 % y una disminución de la sobrestimación de 29.7 % a 28.55 %. Respecto a las otras métricas, el R^2 disminuyó del Modelo 2 al Modelo 3 de 0.431 a 0.420, el error cuadrático medio (MSE) aumentó de 1.7840e+11 a 1.821e+11 y el error absoluto medio (MAE) de 259.051 a 259.686. Ahora, si comparamos el Modelo 2 de las 426 variables originales y el Modelo 4 de las 28 variables seleccionadas por Boruta-XGBoost, vemos que este último no logra alcanzar el porcentaje de éxito del Modelo 2, para ambos intervalos de confianza. Respecto a las métricas MSE y MAE, estas son mayores para el Modelo 4, superando incluso al Modelo 3.

A pesar de obtener resultados similares entre los Modelos 3 y 4 en comparación al Modelo 2 en cuanto a porcentajes de éxito, subestimación y sobrestimación, vemos que a modo general el Modelo 3 es el que mejores métricas tiene con respecto a porcentaje de éxito. No obstante, presenta una disminución en el R^2 y mayores errores MSE y MAE en contraste con el Modelo 2. Por esta razón, se eligió el Modelo 3 (con 35 variables explicativas) como el modelo final, el cual será utilizado para el análisis de interpretabilidad de las predicciones.

IC	Porc. Subestimación	Porc. Éxito	Porc. Sobrestimación
$\pm 15\,\%$	27.80	33.3	38.9
$\pm 25\,\%$	16.8	54.05	29.15

R^2	MSE	MAE
0.405	1.8676e+11	263.108

Tabla 5.2.1: Resultados Modelo 1: 426 variables y parámetros por defecto.

IC	Porc. Subestimación	Porc. Éxito	Porc. Sobrestimación
$\pm 15\,\%$	26.85	34.15	39.0
$\pm 25\,\%$	16.95	53.35	29.7

R^2	MSE	MAE
0.431	1.7840e+11	259.051

Tabla 5.2.2: Resultados Modelo 2: 426 variables y parámetros por Hyperopt.

IC	Porc. Subestimación	Porc. Éxito	Porc. Sobrestimación
$\pm 15\,\%$	26.75	35.15	38.1
$\pm 25\,\%$	17.1	54.35	28.55

R^2	MSE	MAE
0.420	1.8210e+11	259.686

Tabla 5.2.3: Resultados Modelo 3: 35 variables y parámetros por Hyperopt.

IC	Porc. Subestimación	Porc. Éxito	Porc. Sobrestimación
$\pm 15\,\%$	26.4	33.55	40.05
$\pm 25\,\%$	16.3	52.45	31.25

R^2	MSE	MAE
0.412	1.8443e+11	266.571

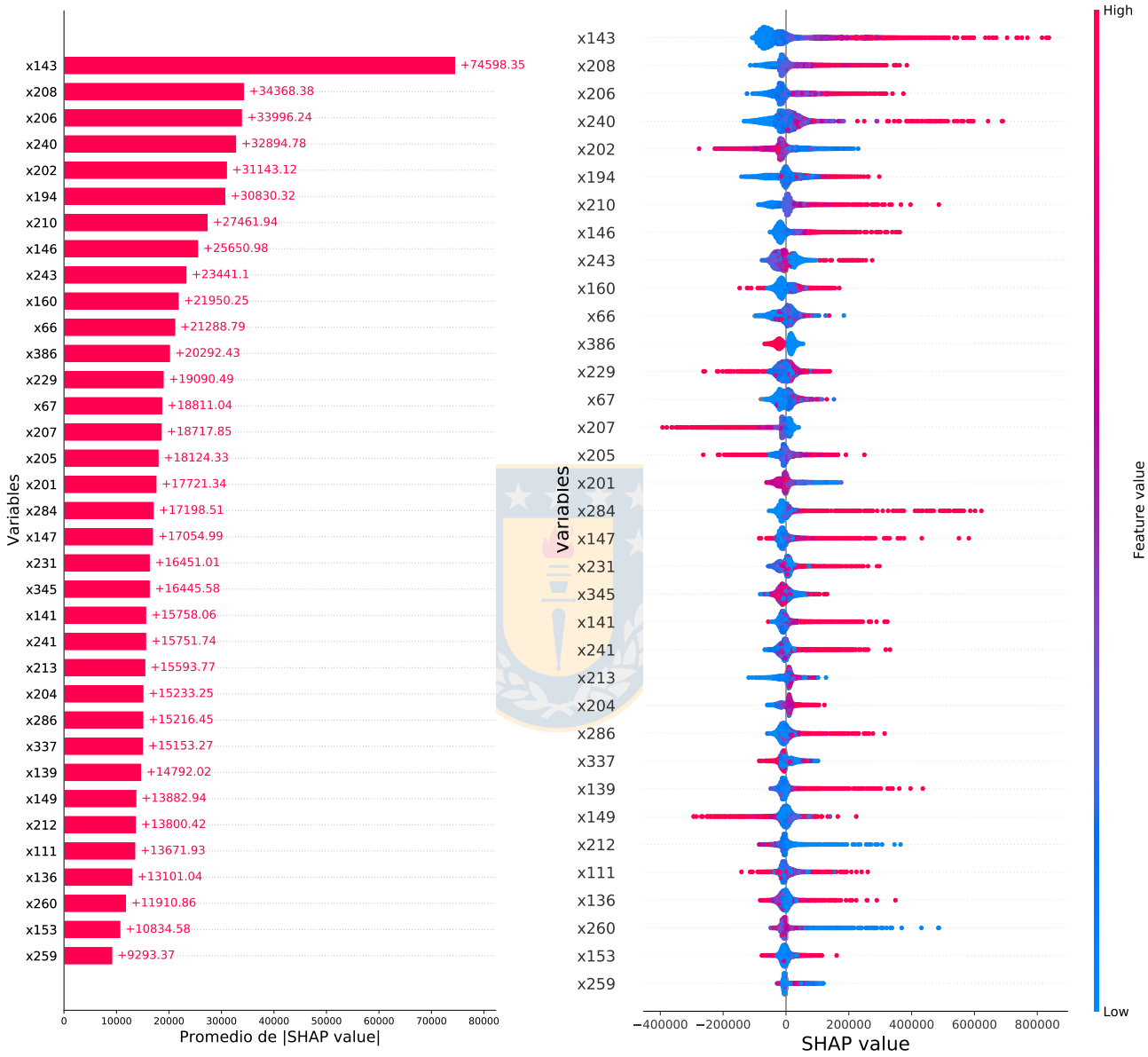
Tabla 5.2.4: Resultados Modelo 4: 28 variables y parámetros por Hyperopt.

5.2.2. Análisis de interpretabilidad a partir del método SHAP

El siguiente análisis se realizó con el modelo XGBoost con 35 variables seleccionadas a partir de BorutaShap (BSHAP), el cual tuvo mejor poder predictivo sobre los otros modelos entrenados. Como interpretabilidad global de las predicciones, primero analizamos el gráfico de barras 5.2.2a que muestra el promedio del valor absoluto de los valores Shapley para cada variable. Inicialmente, observamos que la variable más influyente para el modelo es la $x143$, seguida de las variables $x208$, $x206$ y $x240$. A partir de esta variable en adelante, el promedio de los valores Shapley tiende a estabilizarse. Por otro lado, el SHAP summary plot 5.2.2b se interpreta de la siguiente manera; Dependiendo de los colores que representan el valor de la variable, se podrá ver si dicha variable tiene un efecto positivo o negativo sobre la variable objetivo RentaPromedio. Vemos tal como en el gráfico anterior que la variable más influyente es $x143$, de la cual notamos que a medida que aumenta el valor de esta variable, mayor será la renta estimada de los clientes. Vemos este mismo comportamiento pero en menor magnitud para las variables $x208$, $x206$. Por otro lado, tenemos variables que tienen efecto doble, como la variable $x202$, la cual si aumenta de valor tiene un efecto negativo en la renta estimada, y si disminuye su valor tiene un efecto positivo en la renta estimada, con ambos efectos de igual magnitud. También podemos observar que hay variables que al aumentar su valor tienen un efecto negativo en la renta estimada, como por ejemplo $x207$, $x149$.

En lo que respecta el análisis de interpretabilidad local de las predicciones, se consideraron las predicciones de la renta promedio de cuatro clientes para su análisis: La número #5 (5.2.3), la #474 (5.2.4), la #2412 (5.2.5) y la #5006 (5.2.6). En los 4 gráficos se muestra cómo contribuyen las principales variables en la predicción $f(x)$ de cada observación, a partir del valor promedio $E[f(x)]$. Para la observación #5 vemos que las principales variables que contribuyen negativamente en la predicción son $x202$, $x194$, $x206$, $x207$, $x213$ y $x243$. A través de ellas el modelo predice el valor $f(x) = 744{,}878$ desde el esperado de $E[f(x)] = 904{,}490$. A su vez, vemos que la suma de las contribuciones de las 26 variables restantes es solo de \$7200.81. Para la observación #474 vemos el efecto contrario, ya que la totalidad de las variables tienen un efecto positivo sobre la predicción del modelo, donde la renta promedio de este cliente es de $f(x) = 2{,}339{,}145$. Para la observación #2412 vemos que la renta promedio del cliente es de $f(x) = 1{,}086{,}868$, donde las variables que contribuyen en mayor magnitud con la $x143$ con \$77.244, la $x201$ con \$63.608 y la $x146$ con \$53.103. Finalmente, vemos que para la observación #5006 la mayoría de variables contribuyen negativamente a estimar el

valor $f(x) = 403{,}634$ a partir del promedio $E[f(x)] = 904{,}490$.



(**a**) Gráfico de barras (**b**) Gráfico de resumen

Figura 5.2.2: Interpretabilidad global SHAP

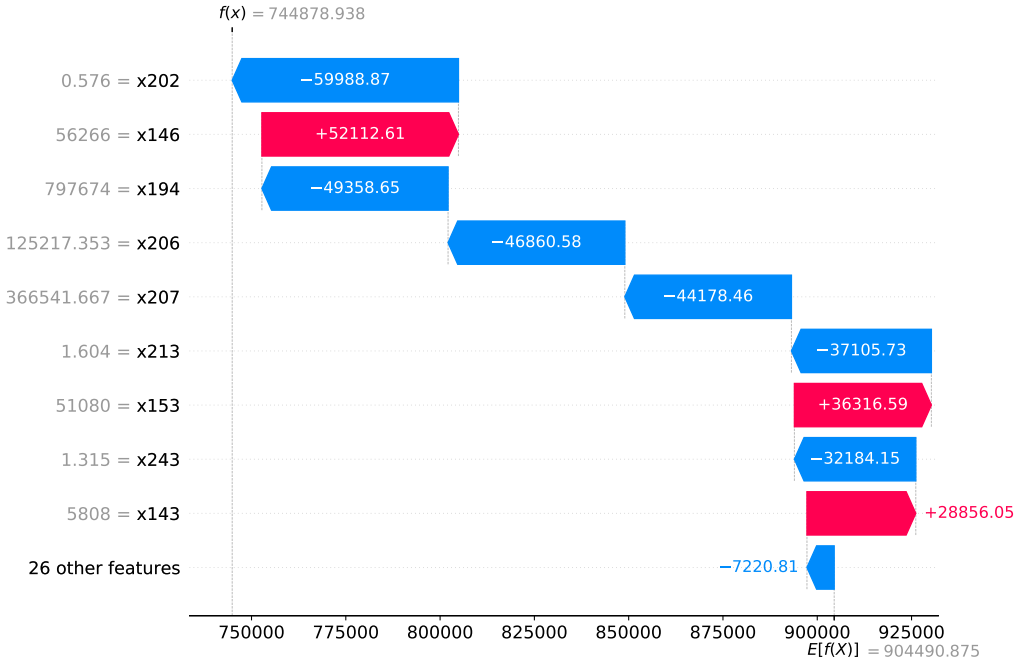


Figura 5.2.3: Explicabilidad local para observación #5

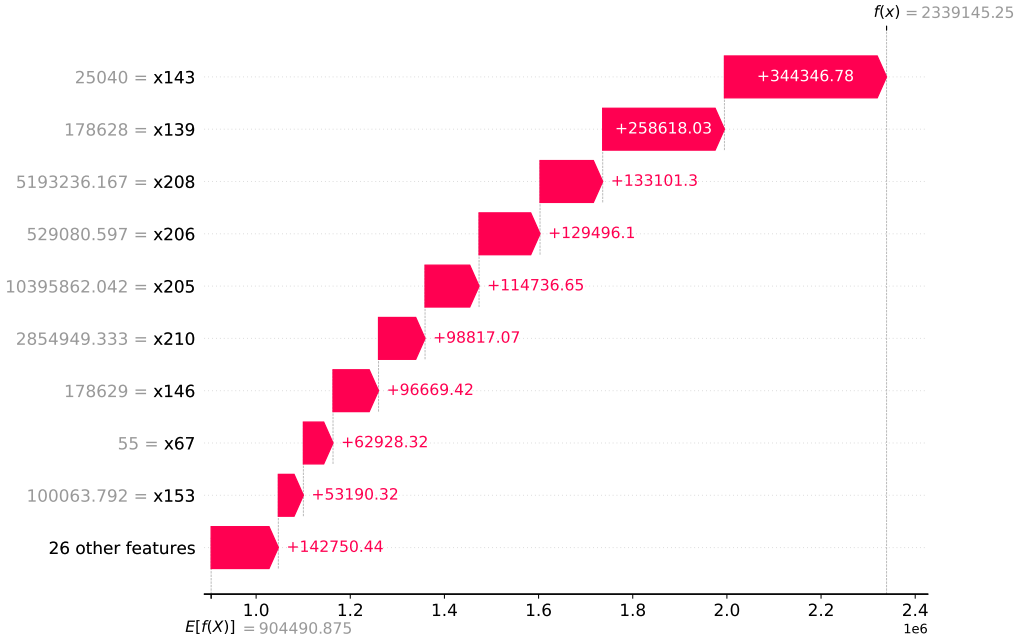


Figura 5.2.4: Explicabilidad local para observación #474

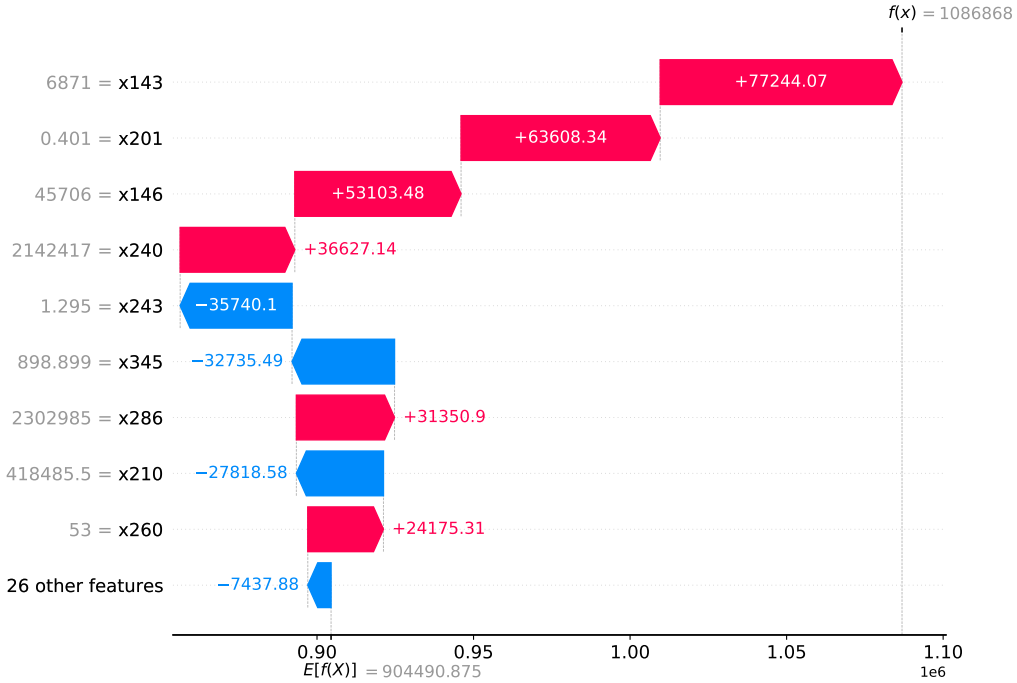


Figura 5.2.5: Explicabilidad local para observación #2412

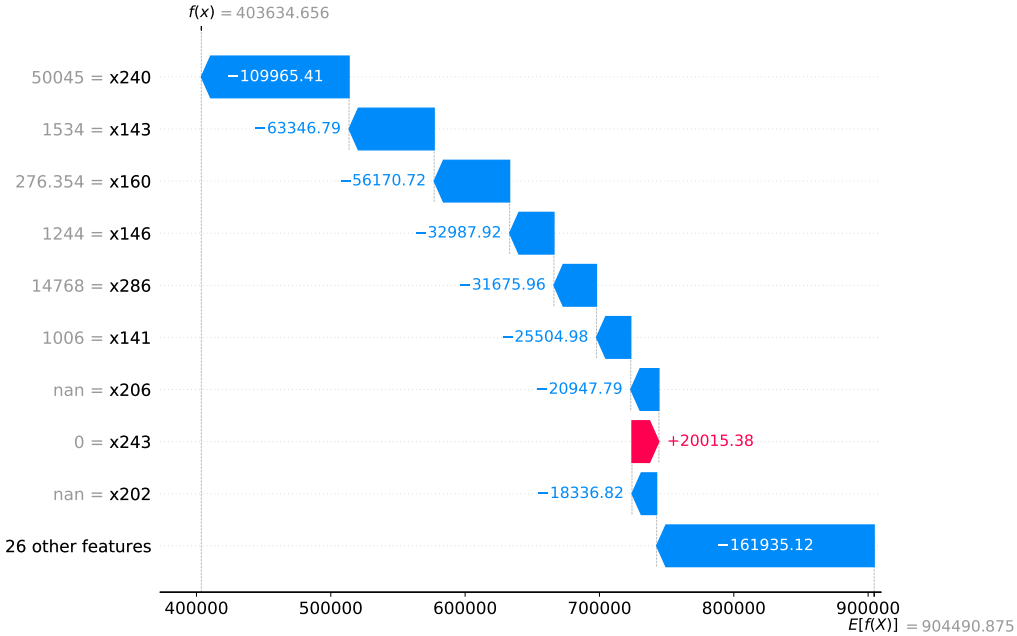


Figura 5.2.6: Explicabilidad local para observación #5006

Capítulo 6

Conclusiones

En este trabajo de tesis se entrenaron modelos XGBoost para estimar la renta de clientes bancarios. También, se abordó un problema de selección de variables, donde se realizó un estudio de simulación con el fin de comparar y validar dos métodos comúnmente utilizados, a saber; Boruta-XGBoost y BorutaShap. Finalmente, al modelo con variables reducidas se le realizó un análisis de interpretabilidad de los resultados mediante la metodología SHAP.

De los resultados obtenidos para el estudio de simulación se concluye que ambos métodos son eficientes al momento de seleccionar un subconjunto de variables importantes o influyentes para el modelo. A pesar de que BorutaShap selecciona de mejor manera las variables influyentes, tiene la desventaja de tomar un tiempo considerablemente mayor cuando se aumenta el número de variables total. Es por esto que en la práctica recomendamos el uso del método Boruta-XGBoost, que tiene una cobertura similar a BorutaShap al momento de seleccionar variables importantes. En este aspecto, nuestro trabajo representa un aporte con una comparación entre dos métodos de selección de variables no abordada aún en la literatura.

En lo que respecta a la aplicación sobre datos reales, podemos concluir que XGBoost es una herramienta versátil para construir modelos de diversos tipos de aprendizaje, debido a su rápida implementación en Python y por la posibilidad de explorar diversas combinaciones de hiperparámetros. Por otra parte, se concluye que el encontrar una combinación óptima de hiperparámetros ayuda a mejorar el poder preditivo (o disminuir el error) de los modelos, en comparación a utilizar los hiperparámetros incluidos por defecto en la librería, en términos de porcentajes de éxito, subestimación y sobrestimación. Por último, el modelo XGBoost entrenado con la base reducida de 35 variables explicativas obtenidas mediante BorutaShap tuvo mejor poder predictivo en cuanto a porcentaje de éxito en comparación a los modelos entrenados con la base original de 426 variables. Esta disminución en la cantidad de variables trae consigo una reducción en la complejidad del modelo, en los tiempos de entrenamiento y

en el almacenamiento de datos, todo esto sin sacrificar el poder predictivo.

Finalmente, con respecto al análisis de interpretabilidad de las predicciones, se utilizó el método SHAP para explicitar como y en qué medida las variables tienen contribuciones sobre las predicciones del modelo. Mediante este análisis, basado en un método con una robusta base teórica, la institución financiera podría ser capaz de conocer la influencia de cada variable sobre las predicciones de renta promedio de los clientes y serán capaces de dar seguimiento a las variables más influyentes.

6.1. Trabajos Futuros

Tal como vimos en este trabajo, los porcentajes de éxito de los modelos entrenados no sobrepasan el 35 % con confianza del $\pm 15\,\%$. A pesar de que estos valores están en rangos aceptables, se pueden desarrollar metodologías basadas en otros algoritmos de ML para aumentarlos.
Por otro lado, debido a la necesidad de contar con modelos explicables en diversas aplicaciones, se propone aplicar nuestra metodología en otras áreas del conocimiento, ya que su uso no está limitado a predecir ingresos de clientes bancarios.

Bibliografía

Alsahaf, A., Petkov, N., Shenoy, V., and Azzopardi, G. (2022). A framework for feature selection through boosting. *Expert Systems with Applications*, 187.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305.

Breiman, L. (2001). Random forests. In *Machine Learning*, pages 5–32.

Chalkiadakis, G., Elkind, E., and Wooldridge, M. (2011). *Computational Aspects of Cooperative Game Theory* (*Synthesis Lectures on Artificial Inetlligence and Machine Learning*). Morgan Claypool Publishers, 1st edition.

Chelgani, S. C. (2021). Estimation of gross calorific value based on coal analysis using an explainable artificial intelligence. *Machine Learning with Applications*, 6:100–116.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA. Association for Computing Machinery.

Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv:1702.08608*.

Dwidarma, R., Permai, S. D., and Harefa, J. (2021). Comparison of logistic regression and xgboost for predicting potential debtors. In *2021 2nd International Conference on Artificial Intelligence and Data Sciences* (*AiDAS*), pages 1–6.

Feng, D. C., Wang, W. J., Mangalathu, S., and Taciroglu, E. (2021). Interpretable xgboost-shap machine-learning model for shear strength prediction of squat rc walls. *Journal of Structural Engineering*, 147(11).

Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., and Yang, G.-Z. (2019). Xai-explainable artificial intelligence. *Science Robotics*, 4(37).

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Husna, N., Bustamam, A., Yanuar, A., Sarwinda, D., and Hermansyah, O. (2020). The comparison of machine learning methods for prediction study of type 2 diabetes mellitus's drug design. volume 2264, pages 03–10.

Keany, E. (2020). Borutashap: A wrapper feature selection method which combines the boruta feature selection algorithm with shapley values.

Kibekbaev, A. and Duman, E. (2016). Benchmarking regression algorithms for income prediction modeling. *Information Systems*, 61:40–52.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324.

Kursa, M. B. and Rudnicki, W. R. (2010). Feature selection with the boruta package. *Journal of Statistical Software*, 36(11):1–13.

Lang, X., Wu, D., and Mao, W. (2022). Comparison of supervised machine learning methods to predict ship propulsion power at sea. *Ocean Engineering*, 245.

Leo Breiman, Jerome Friedman, C. J. S. R. O. (1984). *Classification and Regression Trees*. Chapman and Hall/CRC.

Lundberg, S. and Lee, S.-I. (2017). A unified approach to interpreting model predictions.

Lundberg, S. M., Erion, G. G., and Lee, S.-I. (2019). Consistent individualized feature attribution for tree ensembles.

Marcílio, W. E. and Eler, D. M. (2020). From explanations to feature selection: assessing shap values as feature selection mechanism. In *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 340–347.

Miller, T. (2017). Explanation in artificial intelligence: Insights from the social sciences.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Molnar, C. (2022). *Interpretable Machine Learning*. 2 edition.

Parsa, A. B., Movahedi, A., Taghipour, H., Derrible, S., and Mohammadian, A. K. (2020). Toward safer highways, application of xgboost and shap for real-time accident detection and feature analysis. *Accident Analysis & Prevention*, 136.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Ray, S. (2019). A quick review of machine learning algorithms. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 35–39.

Ridgeway, G. (2005). Generalized boosted models: A guide to the gbm package.

Rozemberczki, B., Watson, L., Bayer, P., Yang, H.-T., Kiss, O., Nilsson, S., and Sarkar, R. (2022). The shapley value in machine learning.

Salas, P., De la Fuente, R., Astroza, S., and Carrasco, J. A. (2022). A systematic comparative evaluation of machine learning classifiers and discrete choice models for travel mode choice in the presence of response heterogeneity. *Expert Systems with Applications*, pages 116–253.

Shapley, L. S. (1953). A value for n-person games.

Shin, S., Austin, P. C., Ross, H. J., Abdel-Qadir, H., Freitas, C., Tomlinson, G., Chicco, D., Mahendiran, M., Lawler, P. R., Billia, F., Gramolini, A., Epelman, S., Wang, B., and Lee, D. S. (2021). Machine learning vs. conventional statistical models for predicting heart failure readmission and mortality. *ESC Heart Failure*, 8(1):106–115.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Wang, F. and Ross, C. L. (2018). Machine learning travel mode choices: Comparing the performance of an extreme gradient boosting model with a multinomial logit model. *Transportation Research Record*, 2672(47):35–45.

Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., and Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40.

Yang, C., Chen, M., and Yuan, Q. (2021). The application of xgboost and shap to examining the factors in freight truck-related crashes: An exploratory analysis. *Accident Analysis Prevention*, 158:106–153.