Predicción de abandonos y pagos en videojuegos Freemium para móviles

by

Alexei Darias Jojorina

A thesis submitted in conformity with the requirements
for the MSc in Economics, Finance and Computer Science

University of Huelva & International University of Andalusia

November 2018

Predicción de abandonos y pagos en videojuegos Freemium para móviles

Alexei Darias Jojorina

Máster en Economía, Finanzas y Computación

Dr. Emilio Congregado Ramírez de Aguilera
Universidad de Huelva y Universidad Internacional de Andalucía

2018

Abstract

« Mobile gaming industry has grown vertiginously at present, and the companies of the sector dedicate a great effort to increase the retention of their users and the monetization. A fundamental aspect for decision making is the prediction of players who abandon and/or make payments within the games. The objective of this research is to predict dropouts (*churn*) and payments in a game of the company Genera Games. For this purpose, the process of Knowledge Discovery in Databases was carried out, from data extraction to pattern recognition. Binary classification methods Logistic Regression, Random Forests and Gradient Boosting were used. Computational processing was done using SQL, R, Python and the statistical analysis software Stata. As results were obtained the predictions for different periods (short, medium and long term), as well as the comparison between the classification methods. »

JEL classification: C38, C51, C52, L82.

Key words: mobile gaming, prediction, churn, payments, binary classification, Logistic Regression, Random Forests, Gradient Boosting.

Resumen

La industria de videojuegos para móviles ha crecido vertiginosamente en la actualidad y las empresas del sector dedican un gran esfuerzo para aumentar la retención de sus usuarios y la monetización. Un aspecto fundamental para la toma de decisiones es la predicción de jugadores que abandonan y/o realizan pagos dentro de los juegos. El objetivo de la presente investigación es predecir abandonos y pagos en un juego de la empresa Genera Games. Para ello se realizó el proceso de Descubrimiento de Conocimiento en Bases de Datos, desde la extracción de datos hasta el reconocimiento de patrones. Se utilizaron los métodos de clasificación binaria Regresión Logística, Bosques Aleatorios y Potenciación del Gradiente. El procesamiento computacional se hizo mediante SQL, R, Python y el software de análisis estadístico Stata. Como resultados se obtuvieron las predicciones para distintos períodos (corto, medio y largo plazo), así como la comparativa entre los métodos de clasificación.

Clasificación JEL: C38, C51, C52, L82.

Palabras clave: videojuegos para móviles, predicción, abandonos, pagos, clasificación binaria, Regresión Logística, Bosques Aleatorios, Potenciación del Gradiente.

Agradecimientos

«A mi familia por el apoyo incondicional y por esas muestras de amor que nunca me han faltado. En especial a mi madre Victoria.

A mi novia Midiala, mi compañera de aventuras. Gracias por ser mi sostén en el día a día, por tu cariño, comprensión y por estar ahí para mí.

A la AUIP y la UNIA, por la oportunidad.

A Emilio, por ser un excelente coordinador y por tanto apoyo.

Al resto de profesores del Máster, profesionales de un altísimo nivel profesional.

A Genera Games, sus directivos, mis compañeros de departamento y todos los que me han hecho parte de la familia.

A mis amigos todos, esos que ahora pasan por mi mente y que no voy a mencionar porque saben quiénes son.

A Cuba y su gente. A España y su gente.

¡Gracias por TANTO!»

Tabla de Contenidos

Lista de Tablas

Lista de Figuras

Lista de Anexos

1 Introducción

A partir de Julio de 2018, se ha gastado más dinero en videojuegos –en todas sus formas y formatos- que en cualquier otro tipo de entretenimiento en el planeta. A día de hoy, la industria del videojuego es un fenómeno masivo que vale casi 3 veces más que la industria del cine (Liftoff, 2018). Durante el año 2017, el mercado de los juegos ingresó aproximadamente 121,7 mil millones de dólares. El crecimiento de la industria se prevé que sea continuo: para 2018 se estima ingresar 137,9 mil millones de dólares, y las predicciones para 2021 son de 180,1 mil millones (Newzoo, 2018).

La industria, que incluye productos para consola, PC y dispositivos móviles, ha presentado un crecimiento vertiginoso en esta última gama. Durante 2017, el 46% del total de ingresos del mercado de videojuegos perteneció a *mobile gaming*. La penetración de este nicho se estima que continúe en ascenso: durante 2018 el 51% de los ingresos pertenecerá a dispositivos móviles, mientras que para 2021, llegará a ser el 59% según estudios (Newzoo, 2018). Por el lado de la demanda, dicha evolución está incentivada por los 3,39 mil millones de usuarios de móviles en todo el mundo y una explosión en el número de momentos de micro-ocio. Por el lado de la oferta, App Annie (proveedor de datos del mercado de aplicaciones) registra la existencia de casi 900.000 juegos en vivo en las tiendas de aplicaciones a nivel mundial: aproximadamente 350.000 en App Store de Apple y 550.000 en Google Play (Liftoff, 2018).

El hecho de ser un negocio fructífero se debe en gran medida al esfuerzo que dedican las empresas del sector a la retención de los usuarios y la monetización.

En *mobile gaming*, la retención describe el porcentaje de usuarios que regresan al juego transcurrido cierto período de tiempo desde la primera sesión jugada. Inversamente proporcional a la retención está la tasa de abandono o *churn rate*: cantidad de individuos que salen de un grupo colectivo (jugadores en este caso), durante un período específico. Las empresas necesitan que los ratios de retención sean altos (*churn rates* bajos), para tener un amplio número de usuarios activos, lo que se traduce en mayor probabilidad de obtener ingresos. Sin embargo, en un escenario de creciente competitividad, se dificulta cada vez más la fidelización de los usuarios. Según reportes estadísticos, durante el primer cuatrimestre de 2018, el ratio de

retención de juegos de acción a siete días fue de 12,1%, mientras que la categoría con mejor porcentaje fue la de juegos de palabras, con 19,6% (Adjust, 2018).

Por otra parte, la monetización es el mecanismo que se implementa en el juego para generar ingresos. Uno de los métodos para monetizar es a través de In-App Purchases (IAP): un modelo en el que se induce al jugador a pagar por oro, potenciadores, accesorios, etc., proporcionándole una mejor experiencia de juego. Sin embargo, no es la mayoría de usuarios los que realizan compras. Está estimado que el costo de llevar a un usuario desde la instalación del juego hasta la IAP es de $28,05 (Liftoff, 2018). A esto se suma todo un mecanismo de impuestos sobre las compras, debido a que en la mayoría de los casos, los ingresos se realizan a través de las tiendas de aplicaciones. En Google Play Store, se establece una comisión por transacción del 30% por cada compra, quedando en manos de la compañía que desarrolló el juego el 70% (Google, 2018). De forma similar, en la App Store de Apple, el impuesto por transacción es del 30% (Viswanathan, 2018). Independientemente de la plataforma que utilicen, las empresas destinan recursos humanos y tiempo a conseguir un sistema de monetización rentable para sus juegos.

Una importante empresa del sector en Sevilla es Genera Games, la cual, junto a Genera Indie Games y otras filiales de equipos *indie*, ha logrado colocar juegos de calidad en el mercado internacional, colaborando con compañías como DreamWorks, Disney, Mattel, Hasbro, Paramount, Lucas Film y La Liga. Los juegos *indie* de Genera usan el modelo Free-to-Play (F2P) o Freemium, es decir, son juegos gratuitos que ofrecen valor añadido por compras dentro de la propia aplicación (IAP). Estos tienen el objetivo de convertirse en productos de gran aceptación por el público y con gran impacto en los ingresos de la empresa.

Tanto en Genera Indie Games como en otras empresas de videojuegos para móviles, existen productos que no consiguen los niveles de retención y monetización deseados. Se trata de juegos que logran llegar a públicos de países considerados *tier 1* en el mercado, como Estados Unidos, sin embargo la retención de nuevos jugadores y el *Gross Margin Per Download* (indicador que expresa la relación entre cantidad ingresada y cantidad de descargas) no son los esperados. Para solventar esta situación, los departamentos de desarrollo, producción, calidad, marketing, arte y analítica, trabajan día a día en nuevas versiones de los productos que impulsen su crecimiento. Ante cada versión a lanzarse cabe plantearse varias interrogantes en cuanto a los indicadores claves de rendimiento (KPI) que tendrá el juego en el futuro, de forma tal que se puedan predecir

estos valores y tomar decisiones puntuales. Además, resulta de interés conocer qué variables influyen en las decisiones de los usuarios de abandonar un juego o realizar compras dentro de éste, para poder incidir en dichas decisiones desde el propio producto.

El presente trabajo surge para darle solución a la situación antes expuesta, por lo que el problema a resolver es: ¿Cómo anticipar las decisiones de abandono y pago por parte de los usuarios de videojuegos Freemium?

La investigación se espera que aporte beneficios para la empresa como son:

- Disponibilidad de métodos evaluados y contrastados para poder predecir, con los datos existentes, el comportamiento de los usuarios en el futuro en cuanto a abandonos y pagos.
- Conocer qué variables son importantes en la interacción de los usuarios con los juegos, para poder explicar las decisiones de abandonar o hacer compras en la aplicación.
- Información útil para la toma de decisiones a nivel de producto, de forma que se disminuya la probabilidad de que un usuario abandone el juego y se aumente la probabilidad de que realice pagos, lo que deriva en un incremento de ingresos.

Además, se pretende que el estudio realizado contribuya a enriquecer el conocimiento existente sobre retención de usuarios y monetización en juegos Freemium para dispositivos móviles.

2 Objetivos

El objetivo principal del trabajo es predecir la ocurrencia de abandonos y pagos en uno de los videojuegos Freemium de Genera Games.

Para dar cumplimiento al objetivo planteado, se precisan las siguientes tareas de la investigación:

1. Realizar una revisión bibliográfica para conocer los antecedentes y el estado actual de la predicción de abandonos y pagos en videojuegos Freemium.

2. Estudiar y asimilar las tecnologías, métodos y técnicas para el análisis de datos en videojuegos, así como estudiar la información que es posible obtener de la interacción jugador-juego.

3. Definir qué se entenderá por usuarios que abandonan y usuarios que hacen pagos para poder predecir dichas variables.

4. Investigar y aplicar una metodología que guíe el proceso de descubrimiento de conocimiento en datos.

5. Investigar y aplicar métodos de clasificación binaria que permitan predecir a los jugadores que abandonarán y/o pagarán.

6. Obtener los resultados de las clasificaciones, realizando ajustes y mejoras a los modelos.

7. Evaluar y comparar los métodos de clasificación, arribando a conclusiones.

3 Antecedentes

Diversos artículos reflejan los resultados de aplicar la predicción de abandonos a negocios con contratación o subscripción: operadoras de telecomunicaciones (Verbeke, Dejaeger, Martens, Hur, & Baesens, 2012), periódicos (Coussemen & Van den Poel, 2008), bancos (Xie, Li, Ngai, & Ying, 2009), servicios de multimedia bajo demanda (Tsai & Chen, 2010), entre otros. En la actualidad, la predicción de abandonos es una de las aplicaciones más populares del Big Data en los negocios, y ha llegado también al sector de los videojuegos.

En (Kawale, Pal, & Srivastava, 2009) se estudió el abandono de jugadores en EverQuest II, un popular Juego de Rol Multijugador Masivo en Línea (MMORPG por sus siglas en inglés). En el trabajo se propuso un modelo de predicción de abandonos basado en examinar la influencia social entre jugadores y su compromiso personal en el juego. Se validó la hipótesis de que la influencia social es una cantidad vectorial, con componentes de influencia negativa y positiva. El factor de influencia social se usó para crear un gráfico de red, utilizando registros de juego grupal.

Por otra parte en (Hadiji, et al., 2014) se definió la predicción de abandonos de usuarios con dos conceptos propios y fueron empleados modelos basados en árboles de decisiones, regresión

logística, redes neuronales y *Naïve Bayes* para su cómputo. Se usó la puntuación F-1 para comparar el desempeño de los modelos, siendo el modelo de árbol de decisión el de mejores resultados.

En (Runge, Gao, Garcin, & Faltings, 2014) se investigó la predicción de abandonos para jugadores de alto valor en juegos sociales casuales, y se proporcionó un marco de trabajo para el análisis de abandonos en juegos. En el artículo se predijo el abandono de un público formado por el 10% de los usuarios más pagadores, en dos juegos casuales para móviles, tipo Freemium. Se usaron cuatro tipos de algoritmos (redes neuronales, regresión logística, árbol de decisiones y máquina de soporte vectorial), y se definió como abandono a un jugador que no jugó durante 14 días consecutivos. El rendimiento de los modelos se evaluó utilizando el área bajo la curva ROC (Característica Operativa del Receptor), conocida por AUC.

En (Kim, Choi, Lee, & Rhee, 2017) se hizo un análisis de abandonos en juegos móviles a partir de los datos de los usuarios. Se definió un período de 5 días para observar a los usuarios, y un período de 10 días para conocer si estos volvieron a jugar. Los algoritmos empleados fueron: regresión logística, bosques aleatorios, potenciación del gradiente y redes neuronales. Se utilizaron AUC y validación cruzada *10-Fold* como medidas del rendimiento.

Los artículos anteriores, si bien ayudan a entender modelos y técnicas para la predicción de abandonos en juegos, no son aplicables a cualquier producto y público objetivo. El trabajo de (Kim, Choi, Lee, & Rhee, 2017), a pesar de ser un estudio intuitivo y aplicable a los objetivos planteados en la presente investigación, establece un período de observación fijo, por lo que los resultados no son extensibles a plazos de observación menores o mayores a 5 días.

En cuanto a los pagos de jugadores, también se han realizado investigaciones enfocadas a la predicción. En (Sifa, et al., 2015) se crearon dos modelos para predecir compras de usuarios de juegos móviles tipo Freemium: un modelo de clasificación para predecir la ocurrencia de la compra, y un modelo de regresión para predecir el número de compras que hicieron los usuarios. Para ambos modelos se utilizaron árboles de decisión/regresión. Según los autores, se trató del primer estudio donde se investigaron las decisiones de compras en juegos Freemium de móviles, desde una perspectiva del comportamiento de los usuarios. Para los modelos se crearon períodos de observación de 1, 3 y 7 días. Los modelos de clasificación fueron evaluados teniendo en cuenta criterios como la precisión y la medida F-1.

En (Guitart, Pei Chen, Bertens, & Periáñez, 2017) se presentó un análisis experimental de varios métodos para pronosticar variables relacionadas con juegos. El trabajo tuvo dos objetivos principales: obtener predicciones de las compras dentro de la aplicación y del tiempo de juego, así como realizar simulaciones de eventos en el juego para maximizar las ventas y el tiempo de juego. Las técnicas empleadas fueron ARIMA (regresión dinámica), potenciación del gradiente, modelos aditivos generalizados y redes neuronales.

Los análisis anteriores ayudan a comprender y formular modelos de predicción de pagos en juegos para móviles. Sin embargo, sus resultados no son extensibles a todos los juegos y jugadores, dada la naturaleza variable e irregular de los videojuegos de móviles y su público.

4 Marco teórico

A continuación se abordarán conceptos relacionados con los datos en videojuegos, estudio que se realizó para comprender cómo se generan, capturan, almacenan, procesan y analizan estos en la industria *mobile gaming*. Además, se definirán los conceptos de usuario que abandona y usuario que hace un pago, para poder hacer las predicciones en base a estas definiciones.

4.1 Datos en videojuegos

En la actualidad, el desarrollo de juegos rentables se ha convertido en un desafío dada la alta competitividad del mercado. Para alcanzar el éxito de los productos, se han desarrollado y/o adoptado herramientas y técnicas provenientes del sector de los negocios, el sector IT y otros. Uno de estos métodos es la analítica, que en los últimos años ha impactado decisivamente en la industria del juego y su investigación.

4.1.1 Game Analytics

La analítica es el proceso de descubrir y comunicar patrones en los datos, en vistas a aportar solución a problemas en negocios o, por el contrario, predicciones para respaldar la toma de decisiones de la empresa, impulsar las acciones y/o mejorar el rendimiento. Los fundamentos metodológicos de la analítica son las estadísticas, la minería de datos, las matemáticas, la programación y la investigación de operaciones, así como la visualización de datos para comunicar los conocimientos adquiridos a las partes interesadas (Drachen, El-Nasr, & Canossa, 2013).

La analítica forma parte de un importante subconjunto de la Inteligencia de Negocios (BI por sus siglas en inglés), en todos los niveles de una empresa u organización, independientemente de su tamaño. BI es un concepto amplio, pero básicamente el objetivo de BI es convertir los datos en bruto en información útil. BI se refiere a cualquier método (generalmente basado en ordenadores) para identificar, registrar, extraer y analizar datos del negocio, ya sea con fines estratégicos u operativos (Watson & Wixom, 2007).

Por su parte, la analítica de juegos o *game analytics* es un dominio de aplicación específico de la analítica, aplicado al contexto del desarrollo e investigación de videojuegos. El beneficio directo que se obtiene al adoptar la analítica de juegos es el soporte para la toma de decisiones en todos los niveles y en todas las áreas de la organización: diseño, arte, programación, marketing, administración e investigación de usuarios. *Game analytics* se dirige tanto al análisis del juego como producto, por ejemplo, si proporciona una buena experiencia de usuario, como al juego como proyecto, es decir, el proceso de desarrollo del juego, incluida la comparación con otros juegos (Drachen, El-Nasr, & Canossa, 2013).

Los datos en *game analytics* son generados a través de telemetría. La telemetría se refiere a datos obtenidos a través de la distancia, comúnmente digitales, pero en principio cualquier señal transmitida es denominada telemetría. En el caso de los juegos digitales, un escenario común es el de un ordenador cliente con el juego instalado, transmitiendo datos de la interacción usuario-juego hacia un servidor de recolección, donde los datos son transformados y almacenados en un formato accesible (Drachen, El-Nasr, & Canossa, 2013).

La presente investigación se centra en los datos que se guardan sobre la interacción de los usuarios con el juego (*user metrics*), aunque cabe destacar que en *game analytics* se estudian otras métricas relacionadas con el rendimiento de la infraestructura técnica y de software del juego, así como las métricas relacionadas con el proceso de desarrollo del videojuego.

Métricas de usuarios

Las métricas de usuarios, también conocidas como métricas de jugador o *user metrics*, son datos relacionados con las personas o usuarios que usan los juegos, desde la doble perspectiva de estos como clientes y como jugadores. La primera perspectiva se centra en el usuario como fuente de ingresos y es usada para calcular métricas relacionadas con ingresos, por ejemplo: el ingreso

promedio por usuario (ARPU por sus siglas en inglés), usuarios activos por día (DAU en inglés), análisis de abandonos, análisis del rendimiento de atención a clientes, análisis de micro-transacciones, etc. La segunda perspectiva es usada para investigar cómo las personas interactúan con el sistema de juego, sus componentes y otros jugadores, enfocándose en el comportamiento *in-game*. Ejemplos de estas métricas son: tiempo total de juego por jugador, promedio de amigos por jugador; y análisis comunes incluyen el estudio de tiempo empleado, de redes sociales, etc. (Drachen, El-Nasr, & Canossa, 2013).

En (Drachen, El-Nasr, & Canossa, 2013) se sugiere una clasificación de las métricas de usuarios en tres grupos:

- Métricas del cliente: Cubren todos los aspectos del usuario como cliente, por ejemplo: el costo de la adquisición de usuarios y la retención. Estas métricas resultan de interés para los profesionales que trabajan con el marketing y la gestión de videojuegos, así como su desarrollo.

- Métricas de comunidad: Cubren los movimientos de la comunidad de usuarios a todos los niveles, por ejemplo: actividad en foros. Estos datos son útiles para, por ejemplo, los *community managers*.

- Métricas de jugabilidad: Cualquier variable relacionada con el comportamiento del usuario como jugador dentro del juego, por ejemplo: la interacción con los objetos y la navegabilidad en el entorno. Estas son las métricas más importantes para evaluar el diseño del juego y la experiencia de usuario. Son útiles para profesionales que trabajan con el diseño, la investigación de usuarios, la calidad, etc.

Relacionado con el último grupo está el análisis de comportamiento. La aplicación de dicho análisis con el propósito de evaluar y comprender el comportamiento de los jugadores ha emergido en los últimos años para convertirse en un componente central del desarrollo comercial y académico de juegos. En los juegos Free-to Play (F2P) o Freemium, los ingresos se deben principalmente a la publicidad y a las compras en el juego. Por tanto, la capacidad de monitorear, analizar y predecir el comportamiento de los jugadores es crucial para construir un negocio sostenible en la industria del juego (Hadiji, et al., 2014).

Por ello la presente investigación está basada en las métricas de jugabilidad para predecir, a partir del comportamiento de los usuarios, las probabilidades de que estos abandonen o realicen pagos dentro de los juegos.

Plataformas de Game Analytics

Para finalmente tener una idea base de la analítica de videojuegos, resulta importante introducir las plataformas tecnológicas que dan soporte a *game analytics*.

Desde el lanzamiento de Dreamcast -la sexta y última consola de videojuegos producida por Sega, lanzada en Estados Unidos el 9 de septiembre de 1999- y el adaptador módem, los desarrolladores de juegos han podido recopilar datos de los jugadores y su comportamiento en el entorno interno (Weber, 2018). Aunque como señala Weber, cabe mencionar que desde un poco antes, con el surgimiento de los primeros títulos online –como EverQuest, en marzo de 1999-, los servidores de juegos almacenaban datos de jugabilidad.

El citado autor expone que la industria del videojuego ha experimentado la evolución de las plataformas de *game analytics* en cuatro etapas:

- Era de archivos planos: los datos se guardan localmente en los servidores de juegos.
- Era de bases de datos: los datos se almacenan en archivos planos y luego se cargan en una base de datos.
- Era de *data lakes*: los datos se almacenan en Hadoop/Amazon S3 y luego se cargan en una base de datos.
- Era sin servidor: se utilizan servicios *gestionados* para el almacenamiento y la consulta de datos.

Específicamente en *mobile gaming* se genera una gran cantidad de datos de telemetría de jugadores y eventos de juegos. La naturaleza de los juegos móviles (muchos dispositivos clientes, conexiones a Internet lentas e irregulares, problemas de batería y administración de energía) hace que la telemetría de los jugadores y el análisis de eventos de juegos se enfrenten a desafíos únicos (Google Cloud, 2018).

Entre los patrones de arquitectura más empleados para analizar los eventos de juegos móviles, -y que recomienda Google-, están el procesamiento en tiempo real de eventos individuales, mediante un patrón de procesamiento de transmisión, y el procesamiento masivo de eventos agregados mediante un patrón de procesamiento por lotes, como se muestra en la siguiente figura:

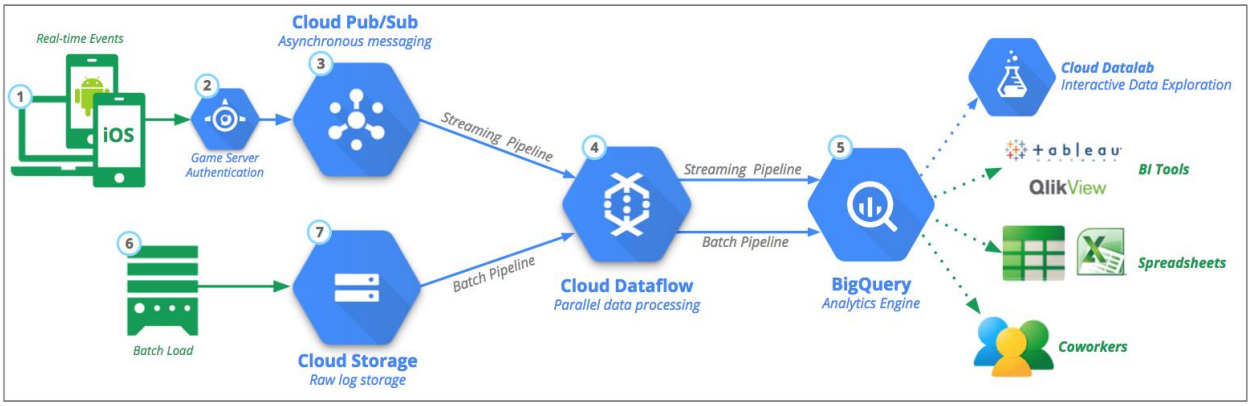

Figura I. Arquitectura de referencia de telemetría de juego. Fuente: *Google Cloud*.

4.2 Definiendo abandonos y pagos

Una vez comprendidos los principales conceptos de *game analytics*, se dio paso a la definición de abandonos y pagos en los juegos, para diseñar los modelos en base a dichas definiciones.

Para estudiar el abandono de usuarios en juegos Freemium, es necesario considerar los aspectos intrínsecos de esta modalidad de juegos. Al ser productos gratis y no contar con un servicio de subscripción o contratación como en las operadoras de telecomunicaciones, periódicos o bancos, no existe un evento que indique con exactitud que el usuario ha abandonado el juego. Podría pensarse a priori en la desinstalación del juego como la acción que pone fin a la relación del usuario con el juego. Sin embargo, es imposible conocer qué usuario ha desinstalado un juego, porque los eventos que son posibles monitorizar son los que se envían desde la aplicación, y no eventos externos como la desinstalación. Por tanto, es común que las empresas del sector definan un período de tiempo en el cual, si el usuario no regresa al juego, se considera que lo ha abandonado.

El hecho de que existan usuarios que retornen al juego a los 3 días de no jugar, a los 7, a los 14 o en el momento que lo decidan hacer, sin que exista una notificación de cuánto no jugarán,

conlleva a la necesidad de establecer un período de tiempo para determinar si el usuario ha realizado un abandono o no. Lo anterior genera el problema de que es posible clasificar como usuarios que abandonan a jugadores que retornan en un futuro lejano, sin embargo, colocar un período de abandono en el infinito no es factible porque nunca se tendría el resultado de la predicción.

Teniendo en cuenta lo anterior, se definieron los siguientes conceptos, que serán los utilizados a lo largo de la investigación para realizar la predicción de abandonos:

- Período de Observación (PO): Cantidad de días en los que se observa el comportamiento de los jugadores, a partir de su primera sesión de juego.

- Período de Abandono (PA): Cantidad de días a partir del PO en los que se observa si los jugadores continúan jugando o no.

- Usuario que abandona (UA): Usuario que no realizó ninguna sesión de juego dentro del PA.

En el caso de los usuarios pagadores, se definieron conceptos similares a los anteriores para poder realizar las predicciones:

- Período de Observación (PO): Cantidad de días en los que se observa el comportamiento de los jugadores, a partir de su primera sesión de juego.

- Período de Pago (PP): Cantidad de días a partir del PO en los que se observa si los jugadores realizan un pago o no.

- Usuario pagador (UP): Usuario que realizó alguna compra dentro del PP. Es necesario aclarar que el hecho de que el usuario haya comprado en el PO no lo convierte, según la definición que se ha empleado, en UP, a menos que también haya comprado en el PP.

Para las estimaciones se seleccionaron períodos de observación, de abandonos y pagos razonables, que van ligados al tipo de análisis que requiere la organización: a corto, medio o largo plazo.

5 Metodología

Una vez comprendida la problemática, establecidos los objetivos de la investigación, y estudiados los antecedentes y el marco teórico, se determinó como siguiente paso la selección del juego sobre el cual se centraría la investigación. Establecido un juego en concreto, se dio inicio al proceso de Descubrimiento de Conocimiento en Bases de Datos (KDD por sus siglas en inglés), es decir, la extracción no trivial de información implícita, previamente desconocida y potencialmente útil de los datos (Frawley, Piatetsky-Shapiro, & Matheus, 1992). De este modo, se siguieron las cinco etapas que se definen en KDD: Selección de los datos, Pre-procesamiento, Transformación, Reconocimiento de patrones y Evaluación e interpretación (Fayyad, Piatetsky-Shapiro, & Smyth, 1996). A continuación se describirán detalladamente las primeras cuatro fases del estudio, haciendo mención de los materiales, técnicas y métodos empleados.

5.1 Selección del juego objetivo

La selección del juego objetivo para la predicción de abandonos y pagadores siguió una serie de requerimientos. Lo primero a tener en cuenta fue la plataforma del juego. A pesar de que en Genera Indie Games existen productos publicados tanto para Android como para iOS, se tomó como referencia un juego de Android. La decisión estuvo fundamentada en obtener una muestra del mercado con mayor número de aplicaciones y por tanto mayores retos de retención: Google Play Store. Este mercado es a su vez el que mayores desafíos enfrenta para conseguir pagos, pues como se ha reportado, los usuarios de iOS son mucho más propensos que los usuarios de Android a gastar en aplicaciones (Liftoff, 2018).

Además, se decidió analizar un producto con una madurez relativa en el mercado, es decir, un videojuego estable. La consola de desarrollador de Google Play Store mide la estabilidad de una App mediante indicadores como: índice de errores ANR (Application Not Responding), que se refiere al porcentaje de sesiones diarias en las que los jugadores han observado al menos un error ANR; tasa de fallos, que es el porcentaje de sesiones diarias en las que los jugadores han observado al menos un bloqueo; y tasa de fallos múltiples, referido al porciento de sesiones diarias en las que los jugadores han observado al menos dos bloqueos. Un juego no estable podría incidir negativamente en la retención de usuarios y la monetización, pero las decisiones de los usuarios de abandonar y/o pagar estarían condicionadas por factores como errores y bloqueos, que no describen el comportamiento normal de un usuario en un juego. Es por ello que

se seleccionó un producto que cumpliese con los límites de comportamiento adecuado que establece Google.

Teniendo en cuenta lo anterior, se seleccionó para el análisis "Bubble Word": un juego casual de palabras, en el que los jugadores deben encontrar palabras uniendo las letras que se le ofrecen en cada nivel. El producto pertenece al género de videojuegos casuales, el cual suele tener reglas simples y está destinado a jugadores no habituales, que buscan una distracción diferente (Hernández Pérez, Cano Gómez, & Parra Meroño, 2016).

5.2 Fuente de datos. Selección de los datos

Como se explicó en el apartado 4.1, en *mobile gaming* se genera una gran cantidad de datos de telemetría de jugadores y eventos de juegos, que sirven de base para realizar *game analytics*. Estos datos son enviados, procesados y almacenados en plataformas, de modo que se pueda extraer información sobre el comportamiento de los jugadores y otra información relevante.

En Genera Indie Games, la plataforma utilizada es deltaDNA: una plataforma de Big Data diseñada para lograr un rápido rendimiento en grandes conjuntos de datos. DeltaDNA utiliza Vertica como almacén de datos y VoltDB como base de datos en memoria en tiempo real. En el corazón de la plataforma se encuentra una combinación de infraestructura de datos NoSQL y NewSQL, que brinda a la plataforma el acceso a tecnologías de base de datos modernas, al mismo tiempo que ofrece a los usuarios acceso a una interfaz SQL estándar para el análisis de sus datos (DeltaDNA, 2018).

En la interfaz de usuario, los datos se distribuyen en ocho tablas principales:

- events: Contiene la información de los eventos generados por los usuarios y varios atributos relacionados con dichos eventos. Se entienden por eventos a las acciones que realizan los usuarios en el juego. Estos son enviados desde el videojuego hasta la plataforma, de modo que se pueda hacer un seguimiento del comportamiento del jugador. Ejemplos de eventos son: inicio de misión, compra de producto, fin de misión, etc.

- fact_user_sessions_day: Contiene agregados de varios atributos para cada sesión del jugador. Se conoce por sesión a cada período de tiempo en el que el usuario abrió el juego y lo cerró durante el día, pudiendo haber varias sesiones por día.

- fact_wau_users: Contiene datos semanales de los jugadores.
- fact_mau_users: Contiene datos mensuales de los jugadores.
- fact_mission_users_day: Contiene detalles de las misiones jugadas, completadas y fallidas para cada jugador, cada día.
- fact_product_users_day: Contiene información de compras de productos para cada jugador, cada día.
- fact_event_type_users_day: Contiene recuentos de tipos de eventos registrados para cada jugador, cada día.
- user_metrics: Contiene agregados de varios atributos por cada usuario.

De estas tablas fue extraída la muestra de usuarios sobre los cuales se realizaron las predicciones de abandonos y pagos. Se utilizó principalmente la tabla fact_user_session_day, por la necesidad de obtener información del comportamiento de los usuarios por día, de modo que se pudiesen obtener los agregados durante el período de observación.

Tras el análisis de los atributos que se guardan para cada usuario, se eligieron todas las variables que pudiesen considerarse, a priori, como parte de los modelos. Estas constituyen las características que describen el comportamiento de los usuarios en el período de observación:

- user_id: Cadena de caracteres. Identificador único asignado a cada usuario de la plataforma.
- sessions: Entero. Cantidad de sesiones jugadas por el usuario.
- revenue: Entero. Cantidad de ingresos que ha generado el usuario, en céntimos de euro.
- missions_started: Entero. Cantidad de misiones que comenzó el usuario.
- missions_completed: Entero. Cantidad de misiones que completó el usuario.
- missions_aborted: Entero. Cantidad de misiones que abandonó el usuario.

- missions_failed: Entero. Cantidad de misiones que falló el usuario.
- total_time_ms: Entero. Cantidad de tiempo, en milisegundos, en que el usuario ha interactuado con el juego.
- transactions: Entero. Cantidad de transacciones (compras) realizadas por el usuario.
- user_country: Cadena de caracteres. País desde el cual juega el usuario.

Para cada usuario se comprobó si en el período de abandono/pago, realizó abandonos/pagos, generando las siguientes variables:

- churn: Binaria. Con valor cero si el usuario jugó durante el período de abandono, o uno si el jugador abandonó, es decir, si no realizó al menos una sesión de juego en ese tiempo.
- payer: Binaria. Con valor uno si el usuario pagó durante el período de pago, o cero si el jugador no realizó ninguna transacción en ese tiempo.

Para elegir una muestra de jugadores homogénea, se seleccionaron nuevos usuarios que comenzaron a jugar la misma versión del juego. Para la muestra se tuvo en cuenta que la versión de juego no cambiase durante los períodos de observación, abandono y pago. Estos filtros resolvieron el problema del ruido que se puede añadir a los modelos, por un comportamiento distinto de los usuarios entre diferentes versiones de la aplicación.

Otro filtro aplicado fue el de usuarios válidos, puesto que en ocasiones se pueden encontrar datos incoherentes en las plataformas. Esto está dado, por una parte, por la naturaleza de la telemetría en *mobile gaming*; como se mencionó anteriormente: gran cantidad de dispositivos clientes, conexiones a Internet lentas e irregulares, problemas de batería y administración de energía. Por otra parte, existen usuarios que utilizan técnicas fraudulentas para *hackear* los juegos y simular, por ejemplo, el pago de una IAP. Para determinar qué usuarios son válidos se comprobaron atributos como la fecha del primer evento, la plataforma utilizada y el atributo de validación de compras.

La muestra resultante se exportó a CSV, formato ampliamente utilizado en el análisis de datos. Dada la limitación de la interfaz web de deltaDNA a la hora de exportar los resultados de

consultas SQL (solo permitió exportar 5.850 tuplas), se tuvo que recurrir a una conexión directa al almacén de datos desde una aplicación de escritorio, usando herramientas compatibles con Postgres ODBC (Open DataBase Connectivity) (DeltaDNA, 2018).

Finalmente para la extracción de datos y su exportación a CSV se utilizó R, entorno de software libre y lenguaje de programación con un enfoque al análisis estadístico. La conexión a la base de datos de deltaDNA fue posible utilizando en RStudio la librería *odbc*, según se indica en la Web oficial de la plataforma (DeltaDNA, 2018). El código fuente para la conexión, así como las consultas SQL para obtener los datos y su exportación a CSV, pueden ser consultados en el Anexo I.

Del proceso de selección de datos resultaron varios data sets, con 72.612 usuarios y 12 variables (10 variables de características, más la variable binaria de abandono/pago). Se extrajeron datos con las siguientes combinaciones en los períodos de observación, abandono y pago:

- Período de Observación = 1, Período de Abandono/Pago = 3: Dataset para el análisis a corto plazo. Se observa el comportamiento de los usuarios durante un día para predecir si durante los siguientes tres días habrán jugando o no, y si harán compras o no. Con los datos del día 1 se puede tener una proyección hasta el día 4.

- Período de Observación = 7, Período de Abandono/Pago = 7: Dataset para el análisis a medio plazo. Se observa el comportamiento de los usuarios durante una semana para predecir si durante la próxima semana habrán jugando o no, y si harán compras o no. Con los datos de la primera semana se puede tener una proyección hasta la semana 2.

- Período de Observación = 14, Período de Abandono/Pago = 16: Dataset para el análisis a largo plazo. Se observa el comportamiento de los usuarios durante dos semanas para predecir si durante dieciséis días después habrán jugando o no, y si harán compras o no. Con los datos de las primeras dos semanas se puede tener una proyección del primer mes.

5.3 Pre-procesamiento de los datos

Luego de seleccionar y extraer los datos, se inició el pre-procesamiento de los mismos. Durante la etapa se realizó un análisis a la muestra para identificar datos inexistentes, datos no clasificados, extremos u *outliers*. En el dataset no hubo presencia de valores nulos o sin

categoría. La mayoría de los atributos estaban en un rango de valores adecuado, sin embargo la variable *total_time_ms* presentó valores en algunos usuarios que no son los esperados ni resultan relevantes para la predicción objetivo.

Se trató de usuarios con un bajo tiempo total de interacción con el juego durante el período de observación. Se debe tener en cuenta que la primera experiencia de un usuario tras lanzar la aplicación (la pantalla de carga inicial), puede influir en su decisión de abandonarla. Sin embargo, no se considera normal que un nuevo jugador haya interactuado menos de un segundo con la aplicación. Una revisión de los usuarios en estos casos evidenció que la gran mayoría presentaba características similares: una única sesión de juego, cero misiones comenzadas/completadas/abandonadas/fallidas y en el campo *churn*, casi todos tenían valor de 1, es decir, no volvieron a jugar. Esta información da a entender que pudieran tratarse de usuarios a quienes el juego no les abrió por problemas técnicos, por lo que se decidió sacarlos de la muestra para eliminar el ruido que pueden aportar.

Luego de la limpieza de los datos, quedaron las muestras distribuidas de la siguiente manera:

- Dataset con P.O = 1: 66.671 usuarios, tras sacar de la muestra a 5.941 usuarios con total_time_ms < 1000.
- Dataset con P.O = 7: 66.839 usuarios, tras sacar de la muestra a 5.773 usuarios con total_time_ms < 1000.
- Dataset con P.O = 14: 66.895 usuarios, tras sacar de la muestra a 5.717 usuarios con total_time_ms < 1000.

El código fuente del pre-procesamiento puede consultarse en el Anexo II.

5.4 Transformación de los datos

Como parte de la transformación de los datos, se crearon variables *dummy* para clasificar a los países de los jugadores según los niveles de ingresos por usuario. Se tomó como referencia la segmentación de países por *tiers* que ofrece Brus Media, empresa de *mobile marketing* con seis años de experiencia en el sector (Brus Media, 2018). Aunque la transformación para nuestros modelos se realizó en tres *tiers*, fusionando los *tiers* 3, 4 y 5 de Brus Media en el *tier* 3. Las

nuevas variables introdujeron a los modelos características ya no solo de comportamiento, sino también demográficas.

Cada dataset se dividió en datos de entrenamiento y datos de validación, escogiendo el 80% de la muestra para entrenar los modelos, en el caso del dataset de abandonos. Como se comentará más adelante, la cantidad de usuarios pagadores fue muy pequeña, por lo que en los dataset de pagadores, los datos de entrenamiento representaron un 70% del total, dejando el 30% para validación. La división fue aleatoria, para ello se creó la variable binaria "train", que tiene valor uno cuando se trata de un jugador del set de entrenamiento, o cero cuando pertenece al set de prueba.

Un vistazo a las variables permitió observar la presencia de atributos continuos, medidos en diferentes escalas y con distintos rangos de valores posibles. En estos casos, es frecuentemente beneficioso escalar todas las características a un rango común, estandarizando los datos (Sudhir Patki & Kelkar, 2013). Los datos fueron normalizados utilizando un método comúnmente aplicado: restando su media y dividiendo por la desviación estándar (Gelman, 2008). Para normalizar el set de entrenamiento, solo se tuvo en cuenta la media y la desviación estándar de los usuarios de este sub-set, garantizando que no se agregase información "futura" si se tomasen la media y la desviación estándar de toda la muestra. Por otra parte, para normalizar los datos de validación, se tomaron la media y la desviación estándar del set conocido (el de entrenamiento). De esta forma se puede probar y evaluar si los modelos pueden generalizarse correctamente a datos nuevos de los que no se tiene toda la información. Para comprobar si la normalización pudo alterar de algún modo los análisis, se contrastaron ambas variantes de los dataset: estandarizados y sin estandarizar.

Las transformaciones anteriores se realizaron mediante el lenguaje de programación Python, el cual ofrece estructuras de datos fáciles de usar y herramientas para el análisis de datos y el trabajo con matrices a través de las librerías "pandas" y "numpy". Además, se utilizó el módulo "sklearn": librerías para realizar aprendizaje automático en Python, que ofrecen herramientas simples y eficientes para la minería de datos y el análisis de datos (Scikit-learn, 2018)

Tentativamente se consideró el uso de nuevas variables de control en forma de ratios. Por ejemplo: la dificultad (relación entre misiones completadas y fallidas) y la frustración (relación entre misiones iniciadas y abandonadas). La idea se descartó debido a la presencia de ceros en

los denominadores. Además, han sido demostradas varias limitantes del uso de ratios para ajustar datos: afecta la distribución de errores de los datos, lo cual puede violar los supuestos de los análisis estadísticos paramétricos subsiguientes; pueden introducir correlaciones espurias entre ratios y otras variables; pueden crear dificultades interpretativas, etc. (Allison, Paultre, Goran, Poehlman, & Heymsfield, 1995).

5.4.1 Análisis de datos "raros" o minoritarios. Balanceo de la muestra

Otra fase importante durante la transformación previa al reconocimiento de patrones, fue el análisis de las muestras para detectar clases "raras" o minoritarias.

Gran parte de la investigación sobre la "rareza" en los datos, se refiere a "clases raras" o, más generalmente, al desbalanceo de clases. Este tipo de rareza requiere ejemplos etiquetados y está asociado con problemas de clasificación. Un segundo tipo de rareza se refiere a "casos raros". Informalmente, los casos raros corresponden a un subconjunto significativo pero relativamente pequeño de los datos. Los casos raros dependen solo de la distribución de datos y, por lo tanto, se definen para datos etiquetados y no etiquetados, y para tareas de minería de datos supervisadas y no supervisadas (Weiss, 2004).

Se habla de "rareza absoluta", cuando el número de ejemplos asociados con las clases/casos raros es pequeño en un sentido absoluto. Por otra parte, la "rareza relativa" está presente en objetos que no son raros en un sentido absoluto, pero que son raros en relación a otros objetos. Si una clase o caso cubre el 1% de una muestra de mil entradas, se puede decir con certeza que la clase/caso es rara(o). Sin embargo no se puede asumir lo mismo para una muestra con 10 millones de observaciones (Weiss, 2004).

Al analizar los datos extraídos en la fase anterior, respecto a las clases de usuarios que abandonan o pagan, se aprecian dos comportamientos diferentes. En los data sets de usuarios que hacen *churn,* con períodos de observación de uno, siete y catorce días, la clase *churn* estuvo presente, respectivamente, en el 61,7%, 76,1% y 81,1% del total de las muestras, por lo que no hay presencia de clases minoritarias. Sin embargo, en los data sets de usuarios pagadores, con períodos de observación de uno, siete y catorce días, la clase *payer* estuvo presente, respectivamente, en el 0,4%, 0,3% y 0,3% del total de las muestras. Esta variable dependiente representa una clase rara o minoritaria.

El problema fundamental de las clases minoritarias está asociado a la falta de datos. En esta situación, la ausencia de datos dificulta la detección de regularidades dentro de las clases/casos raros. Se ha analizado el impacto que los casos raros tienen en el rendimiento de la clasificación. Un estudio, que empleó conjuntos de datos generados sintéticamente, mostró que los casos raros tienen una tasa de errores de clasificación mucho más alta que los casos comunes (Weiss, 2004).

Varios son los métodos que se pueden aplicar para tratar el desbalanceo de los datos, por ejemplo: sobre-muestreo de las clases minoritarias, sub-muestreo de las clases comunes, entrenar solo con las clases raras, segmentar los datos, etc. Una recomendación que se ofrece en (Weiss, 2004), es la de tratar todos los datos de entrenamiento disponibles, sin importar la distribución resultante de clases (o casos). De esta forma, no se descartan datos y no se pierde información.

En este sentido, la estrategia que se siguió para hacer un rebalanceo de los usuarios pagadores, fue la de sobre-muestreo. Se intentaron inicialmente técnicas de sobre-muestreo avanzadas, como SMOTE (*Synthetic Minority Over-sampling Technique*), la cual introduce nuevos ejemplos no replicados de la clase minoritaria, agregando ejemplos de los segmentos de línea que unen a los *k-vecinos* más cercanos a la clase minoritaria. El principal problema de su uso fue que las variables de la muestra, que son enteras, fueron transformadas a continuas en las nuevas observaciones generadas. Aunque el método SMOTE implementado para Python no maneja data sets donde todas las características son nominales, fue generalizado a data sets con una mezcla de variables continuas y nominales: SMOTE-NC (SMOTE-Nominal Continuos) (Chawla, Bowyer, Hall, & Kegelmeyer, 2002). Al intentar esta variante, especificando a *total_time_ms* y *revenue* como variables continuas, tampoco se logró un resultado satisfactorio en cuanto a la distribución de los datos en las nuevas observaciones, por lo que fue descartado su uso. Se decidió utilizar métodos de sobre-muestreo más simples, teniendo en cuenta que en la revisión bibliográfica se explica que algunos estudios han demostrado que el sobre-muestreo es ineficaz para mejorar el reconocimiento de la clase minoritaria, sin embargo, otros estudios han llegado a la conclusión opuesta (Weiss, 2004), por lo que se comprobará su utilización.

El proceso de sobre-muestreo solo se ejecutó en los datos de entrenamiento. De esta forma se garantizó que no entrasen al set de entrenamiento observaciones repetidas en el set de validación. Tras el sobre-muestreo se generó una muestra en el que aproximadamente el 25% de usuarios eran pagadores. Lo anterior se logró a través del uso de Python y los módulos de *sklearn*.

Específicamente, se utilizó el método *resample*, que hace re-muestreo de arreglos o matrices de una manera consistente, cuya estrategia predeterminada implementa un paso del procedimiento de *bootstrapping*.

El código fuente de los procesos de transformación puede consultarse en el Anexo II.

5.5 Reconocimiento de patrones

En la etapa de reconocimiento de patrones (también conocida por minería de datos), se seleccionaron y justificaron tres tipos de modelos de clasificación binaria para hacer las predicciones. Se computó cada tipo de modelo sobre los distintos data sets, según sus períodos de observación y abandono/pago específicos, y teniendo en cuenta la variable a predecir. Además se hicieron análisis con los datos normalizados y sin normalizar, y en el caso de los pagos, se tuvieron en cuenta los datos con y sin sobre-muestreo. En total, se obtuvieron resultados de un total de 54 predicciones. Para un mejor entendimiento, se muestra el siguiente esquema sobre los análisis hechos con el dataset con Período de Observación = 1 y Período de Abandono / Período de Pago = 3:

Abandonos (6 variantes):

Modelo A:
- Datos normalizados
- Datos no normalizados

Modelo B:
- Datos normalizados
- Datos no normalizados

Modelo C:
- Datos normalizados
- Datos no normalizados

Pagos (12 variantes):

Modelo A:
- Sobre-muestreo y datos normalizados
- Sobre-muestreo y datos no normalizados
- Sin sobre-muestreo y datos normalizados
- Sin sobre-muestreo y datos no normalizados

Modelo B:
- Sobre-muestreo y datos normalizados
- Sobre-muestreo y datos no normalizados
- Sin sobre-muestreo y datos normalizados
- Sin sobre-muestreo y datos no normalizados

Modelo C:
- Sobre-muestreo y datos normalizados
- Sobre-muestreo y datos no normalizados
- Sin sobre-muestreo y datos normalizados
- Sin sobre-muestreo y datos no normalizados

5.5.1 Modelos de clasificación binaria

Como se ha mencionado anteriormente, el objetivo del presente trabajo es predecir las decisiones de los usuarios de abandonar o no un juego, o realizar o no una IAP, constituyendo problemas de clasificación binaria.

Varios son los métodos de clasificación binaria que se han aplicado a la predicción de abandonos, pagos y otros tipos de predicciones binarias, como se comentó en el acápite de antecedentes. En el presente trabajo se utilizaron tres algoritmos tradicionalmente empleados en el campo, según se observó en la revisión bibliográfica: Regresión Logística, *Random Forests* o Bosques Aleatorios y *Gradient Boosting* o Potenciación del Gradiente.

5.5.1.1 Regresión Logística

Los objetivos del modelo de regresión logística son principalmente tres: determinar la existencia o ausencia de relación entre una o más variables independientes "X" y una variable dependiente dicotómica "Y". En segundo lugar, medir el sentido (signo) de dicha relación, en caso de que exista. Como tercer objetivo está estimar o predecir la probabilidad de que se produzca el suceso o acontecimiento definido como "Y = 1", en función de los valores que adoptan las variables independientes (Salas Velasco, 1996).

Dado un modelo para explicar una decisión binaria a partir de una única variable independiente:

$$Y = \propto + \beta X + u$$

donde $\propto$ es el término independiente o constante, β es el coeficiente de regresión asociado a la variable independiente X, y u es el término de perturbación aleatoria. Si se quiere determinar la probabilidad P(Y=1), es decir, la probabilidad de que se tome la decisión, el modelo que mejor estima tal probabilidad, debido a que restringe los valores predichos a su rango natural [0,1] es (Salas Velasco, 1996):

$$P = \frac{e^{(\alpha+\beta X)}}{1 + e^{(\alpha+\beta X)}} = \frac{1}{1 + e^{-(\alpha+\beta X)}}$$

A esta expresión se le conoce como función logística, y puede expresarse también de la siguiente forma:

$$\frac{P}{1-P} = e^{(\alpha+\beta X)}$$

Tomando logaritmos neperianos (Ln), la función logística se puede transformar en una función lineal:

$$Ln\left(\frac{P}{1-P}\right) = (\alpha + \beta X)$$

Haciendo otras transformaciones matemáticas, la función logística conduce a la expresión:

$$Ln\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = Ln\left(\frac{P(Y=1)}{P(Y=0)}\right)$$

Donde el cociente P(Y=1)/P(Y=0) se conoce como *odds*, es decir, la razón entre la probabilidad de que se produzca un acontecimiento o suceso, y la probabilidad de que no se produzca. Esta medida se puede utilizar para valorar y estimar la magnitud de la ocurrencia del evento. La *odds* admite valores que van desde 0, cuando P(Y=1)=0, hasta infinito, cuando P(Y=1)=1. (Salas Velasco, 1996).

El logaritmo de la *odds* se conoce como *logit* (L): L=Ln(odds)=Ln[P/(1-P)]. Los posibles valores de L pueden oscilar entre $-\infty$, si P(Y=1)=0 y $+\infty$, si P(Y=1)=1. Los *logits* son funciones lineales de las variables independientes, mientras que la probabilidad estimada P(Y=1) es una función curvilínea, en forma de "S". La estimación sigmoidea, y por tanto no lineal, permite que las estimaciones de las probabilidades predichas se mantengan en el rango de valores comprendidos entre 0 y 1 (Salas Velasco, 1996).

En el modelo multivariable (más de una variable explicativa), durante el proceso de estimación de los valores de los parámetros poblacionales por el método de máxima verosimilitud, se adoptan ciertos criterios: los valores estimados por la función logística, o valores de los *logit* de Y, se relacionan de forma lineal con las variables explicativas X_i. En cambio, la *odds* y las probabilidades estimadas no son funciones lineales. Otros criterios asumidos en la construcción de modelos de regresión logística multivariable son: la independencia entre observaciones y la ausencia de error en la medida de las variables (Salas Velasco, 1996).

Las predicciones de la presente investigación se basan en el modelo multivariable, donde, a priori:

Variables independientes $X_i = \begin{bmatrix} sessions \\ revenue \\ \dots \\ tier3 \end{bmatrix}$; Variable dependientes Y_1: probabilidad de que el usuario abandone el juego, Y_2: probabilidad de que el usuario realice una compra.

5.5.1.2 Bosques Aleatorios (Random Forests)

Random Forests (o *Random Forest*) es una técnica basada en la generación de árboles de clasificación o regresión, haciendo re-muestreo (*bootstrap*) de los datos de diseño, y por otro lado, muestreo de las variables de entrada en cada nodo del árbol. Para una mejor comprensión del algoritmo, se hace necesario introducir brevemente los métodos estadísticos basados en árboles, pasando por los árboles de clasificación y el método *bagging*.

Los métodos estadísticos basados en árboles engloban a un conjunto de técnicas supervisadas no paramétricas, que consiguen segmentar el espacio de los predictores en regiones simples, dentro de las cuales es más sencillo manejar las interacciones (Amat Rodrigo, 2017). Son ampliamente utilizados en muestras con múltiples variables, que interactúan entre ellas de forma no necesariamente lineal. Aunque los métodos basados en árboles pueden ser utilizados tanto para regresión como para clasificación, en el trabajo nos hemos centrado en la clasificación.

El proceso de construcción de un árbol de predicción se divide en dos etapas: división sucesiva del espacio de los predictores, generando regiones no solapantes (nodos terminales), y predicción de la variable a explicar en cada región. Para decidir en dónde se realizan las divisiones, se recurre a la división binaria recursiva (*recursive binary splitting*). Existen varios métodos para encontrar los nodos más puros/homogéneos sobre los cuales realizar las divisiones en los árboles de clasificación (Amat Rodrigo, 2017).

Tras la creación del árbol de clasificación, las observaciones de entrenamiento quedan agrupadas en los nodos terminales. Para predecir una nueva observación, se recorre el árbol en función del valor de sus predictores hasta llegar a uno de los nodos terminales. Suele emplearse la moda de la variable a explicar como valor de predicción, es decir, la clase más frecuente del nodo (Amat Rodrigo, 2017).

Por su parte, el método *bagging* (diminutivo de *bootstrap aggregation*), hace referencia al empleo del muestreo repetido (*bootstrapping*) con el fin de reducir la varianza de algunos métodos de aprendizaje estadístico, entre ellos los árboles de predicción. Dadas *n* muestras de observaciones independientes $Z_1,\ldots, Z_n$, cada una con varianza σ^2, la varianza de la media de las observaciones es σ^2/n. En otras palabras, promediando un conjunto de observaciones se reduce la varianza. Por lo tanto, una forma de reducir la varianza y aumentar la precisión de un método predictivo es obtener múltiples muestras de la población, ajustar un modelo distinto con cada una de ellas, y hacer la media de las predicciones resultantes. En la práctica no se suele tener acceso a múltiples muestras de entrenamiento, pero se puede hacer *bootstrap*, tomando repetidos ejemplos del set de entrenamiento (James, Witten, Hastie, & Tibshirani, 2013). Con este enfoque, se generan pseudo-muestras con las que ajustar diferentes modelos y después agregarlos. A este proceso se le conoce como *bagging* y es aplicable a una gran variedad de métodos de regresión (Amat Rodrigo, 2017), por ejemplo, a los árboles de predicción.

Un aspecto mejorable del método *bagging* es la poca reducción de varianza que se consigue en data sets donde existe un predictor muy fuerte. En estos casos, al aplicar *bagging* a árboles, la mayoría de árboles creados usan el mismo predictor para la división, por lo que son muy parecidos entre ellos. Esto trae por consecuencia que las predicciones de estos "*árboles bagging*" estén altamente correlacionadas (James, Witten, Hastie, & Tibshirani, 2013).

Los árboles aleatorios (*Random Forests*) proporcionan una mejora sobre "*árboles bagging*", por medio de un pequeño ajuste que reduce la correlación a los árboles. Al igual que en el *bagging*, se construyen árboles de decisión en muestras de entrenamiento obtenidas mediante *bootstrap*. Pero cuando se construyen los árboles de decisión, cada vez que se considera una división en un árbol, se elige una muestra aleatoria de *m* predictores como candidatos de división, del conjunto completo de *p* predictores. La división puede usar solo uno de esos *m* predictores. De esta forma, se toma una nueva muestra de *m* predictores en cada división, y normalmente se elige $m \approx \sqrt{p}$, es decir, el número de predictores considerados en cada división es aproximadamente igual a la raíz cuadrada del número total de predictores (James, Witten, Hastie, & Tibshirani, 2013).

Los bosques aleatorios superan el problema de *bagging* con la alta correlación, obligando a cada división a considerar solo un subconjunto de los predictores. Por lo tanto, en promedio $(p-m)/p$ de las divisiones ni siquiera consideran al predictor fuerte, por lo que otros predictores tendrán

más posibilidades. Se puede pensar en este proceso como una *decorrelación* de los árboles, lo que hace que el promedio de los árboles resultantes sea menos variable y, por tanto, más confiable (James, Witten, Hastie, & Tibshirani, 2013).

Para una definición formal se puede decir que un bosque aleatorio es un clasificador que consiste en una colección de clasificadores con estructura de árbol $\{h(x, \Theta_k), k = 1,...\}$ donde $\{\Theta_k\}$ son vectores aleatorios independientes e idénticamente distribuidos y cada árbol emite un voto unitario para la clase más popular en la entrada x (Breiman, 2018).

El algoritmo de bosque aleatorio (tanto para clasificación como para regresión), es el siguiente (Liaw & Wiener, 2002):

1. Dibujar n_{tree} muestras *bootstrap* de los datos originales.

2. Para cada una de las muestras, desarrollar una clasificación sin afinar o árbol de regresión, con la siguiente modificación: en cada nodo, en lugar de elegir la mejor división entre todos los predictores, de forma aleatoria obtener una muestra m_{try} de los predictores y elegir la mejor división entre esas variables. (*Bagging* puede considerarse como el caso especial de bosques aleatorios obtenidos cuando $m_{try} = p$, el número de predictores).

3. Predecir los nuevos datos agregando las predicciones de los n_{tree} árboles (la mayoría de votos para la clasificación y el promedio para la regresión).

Afinando hiperparámetros. Validación cruzada.

Uno de los puntos a tener en cuenta en bosques aleatorios son los hiperparámetros a utilizar para optimizar el rendimiento del método, los cuales deben ser establecidos antes de entrenar. En el caso de un bosque aleatorio, los hiperparámetros incluyen el número de árboles de decisión en el bosque, el número de características consideradas por cada árbol al dividir un nodo, el máximo de niveles en cada árbol de decisión, el mínimo de puntos de datos ubicados en un nodo antes que el nodo sea dividido, entre otros.

Un procedimiento estándar para la optimización (*tuning*) de hiperparámetros, teniendo en cuenta el sobre-entrenamiento que se puede generar en los datos de entrenamiento, es la validación

cruzada, siendo *K-Fold CV* un método comúnmente utilizado (Koehrsen, 2018). En *K-Fold CV*, se divide el conjunto de entrenamiento en *K* subconjuntos, llamados pliegues. Luego se ajusta iterativamente el modelo *K* veces, cada vez entrenando los datos en *K-1* de los pliegues y evaluando en el pliegue *K*. Al final del entrenamiento, se promedia el rendimiento en cada uno de los pliegues para obtener las métricas de validación finales para el modelo.

Para la afinación de hiperparámetros, se pueden realizar varias iteraciones de todo el proceso *K-Fold CV*, cada vez utilizando diferentes configuraciones en el modelo. Luego se comparan todos los modelos, se selecciona el mejor, y se entrena con este todo el conjunto de entrenamiento, para luego evaluar en el set de pruebas (Koehrsen, 2018).

5.5.1.3 Potenciación del Gradiente (Gradient Boosting)

Gradient Boosting es otra técnica de aprendizaje automático utilizada para el análisis de regresión y para problemas de clasificación estadística. La técnica es muy diferente a *Random Forest*, porque no se basa en tanteos o muestreos de las observaciones. Al contrario, *Gradient Boosting* busca una función de predicción óptima, minimizando una función de pérdida (*loss function*) mediante un procedimiento de descenso del gradiente.

El método de potenciación del gradiente emplea la técnica de *boosting*, es decir, ajusta secuencialmente múltiples modelos sencillos, llamados *weak learners*, de forma que cada modelo aprende de los errores del anterior. Como valor final, se toma la media de todas las predicciones (en variables continuas), o la clase más frecuente (en variables cualitativas) (Amat Rodrigo, 2017).

Gradient Boosting implica tres elementos fundamentales: la función de pérdida a optimizar, un algoritmo "débil" para hacer las predicciones, y un modelo aditivo para añadir los *weak learners* que minimizan la función de pérdida (Brownlee, 2016).

La función de pérdida depende del tipo de problema a resolver, aunque esta debe ser diferenciable. Para la clasificación se pueden emplear la pérdida logarítmica y la pérdida exponencial (Schapire, 2003). Un beneficio de la potenciación del gradiente es que no se tiene que derivar un nuevo algoritmo de *boosting* para cada función de pérdida que se quiera usar, sino que es una técnica suficientemente genérica en la que puede usarse cualquier función de pérdida diferenciable (Brownlee, 2016).

Gradient Boosting utiliza para su algoritmo "débil" a los árboles de decisión. Los árboles se construyen de manera codiciosa, eligiendo los mejores puntos de división según las puntuaciones de pureza, o para minimizar la pérdida. Es común restringir a los *weak learners* con parámetros específicos, como un número máximo de nodos, divisiones, o nodos hoja. Esto permite asegurar que los algoritmos permanezcan débiles, pero que aún puedan ser construidos de manera codiciosa (Brownlee, 2016).

En cuanto al modelo aditivo, los árboles se agregan de uno en uno y los árboles existentes en el modelo no se modifican. Se utiliza un procedimiento de descenso de gradiente para minimizar la pérdida al agregar árboles. Después de calcular la pérdida, para realizar el procedimiento de descenso del gradiente, se agrega un árbol al modelo que reduzca la pérdida (es decir, se sigue el gradiente). La salida para el nuevo árbol se agrega luego a la salida de la secuencia existente de árboles, en un esfuerzo por corregir o mejorar la salida final del modelo (Brownlee, 2016).

Para la ejecución del algoritmo se define un número fijo de árboles, o el entrenamiento se detiene una vez que la pérdida alcanza un nivel aceptable o ya no mejora en un conjunto de datos de validación.

La técnica de potenciación del gradiente tiene ventajas como son el manejo natural de datos de tipo mixto (características heterogéneas) y la robustez frente a valores atípicos en el espacio de salida (mediante funciones de pérdida robusta). Una de sus principales desventajas es que la escalabilidad, debido a la naturaleza secuencial del *boosting*, difícilmente puede ser paralelizada (Scikit-learn, 2018).

Afinando hiperparámetros. Validación cruzada.

De forma similar a lo descrito al final del apartado de *Random Forest*, los resultados de *Gradient Boosting* son sensibles a los hiperparámetros con los que se entrenan los modelos. Estos pueden ser contrastados mediante validación cruzada, por ejemplo, utilizando el método *K-Fold*. De esta forma, es posible obtener el set de hiperparámetros que mejora el poder predictivo del modelo en los datos de entrenamiento. La elección del mejor conjunto de hiperparámetros se basa en el criterio de evaluación que resulte de interés, por ejemplo, la precisión.

5.5.2 Criterios de evaluación

Como criterio para evaluar el poder predictivo de los algoritmos se consideraron inicialmente el área bajo la curva ROC, también conocida por AUC, y la precisión del modelo.

La curva ROC (acrónimo en inglés de Característica Operativa del Receptor) se utilizó por primera vez en la teoría de detección de señales para representar el intercambio entre las tasas de aciertos y las tasas de falsas alarmas. El uso del gráfico se ha extendido a la comparación y evaluación de algoritmos de aprendizaje automático. La curva ROC puede ser interpretada como la razón entre el ratio de predicciones verdaderas positivas y el ratio de predicciones falsas positivas, según varía el umbral de discriminación (valor a partir del cual se clasifica a una clase como positiva). El área bajo la curva ROC, o simplemente AUC, proporciona un buen "resumen" para el rendimiento de las curvas ROC (Huang & Ling, 2005).

Por otra parte, la precisión puede ser definida como: $\frac{TP \cdot n_0 + TN \cdot n_1}{n_0 + n_1}$, donde TP es el ratio de verdaderos positivos, TN es el ratio de verdaderos negativos, n_0 y n_1 el número de casos positivos y negativos respectivamente.

En estudios realizados se han establecido criterios precisos y objetivos para comparar ambas medidas en general y se demuestra, tanto empíricamente como formalmente, que el AUC es una medida mejor que la precisión (Huang & Ling, 2005).

Es por ello que se decidió utilizar el criterio AUC sobre los datos de validación, para evaluar el poder predictivo de los métodos empleados. Se adoptaron los rangos de exactitud de clasificación que se establecen en (Zhu, Zeng, & Wang, 2010): 0.9 < AUC <1.0: “Excelente”, 0.8 < AUC < 0.9: “Bueno”, 0.7 < AUC < 0.8 “Sin valor” y AUC < 0.7 “Malo”.

Para reforzar la evaluación mediante el AUC, se calculó en cada caso la precisión de clasificaciones positivas y negativas.

6 Resultados

A continuación se muestran los resultados de las predicciones de abandonos y pagos en los conjuntos de datos extraídos. Por cada método se exponen las herramientas computacionales utilizadas, así como los parámetros que mejoraron el área bajo la curva ROC (AUC). Por último,

se realiza una comparativa general de todos los algoritmos. Las estimaciones fueron ejecutadas sobre los datos de entrenamiento y evaluadas en los datos de validación.

6.1 Regresión Logística

Las regresiones logísticas fueron ejecutadas por medio del paquete de software estadístico Stata, a través del comando *logit*. Este comando ajusta un modelo *logit* para una respuesta binaria por máxima verosimilitud; modela la probabilidad de un resultado positivo dado un conjunto de regresores. La variable dependiente, distinta de cero y no nula (normalmente igual a uno) indica un resultado positivo, mientras que la variable dependiente igual a cero indica un resultado negativo (Stata, 2018).

Para los modelos estimados las variables dependientes fueron: *churn* (para predecir abandonos) y *payer* (para predecir pagadores). Las variables independientes fueron, a priori, aquellas extraídas y transformadas en las fases previas al reconocimiento de patrones, las cuales describen el comportamiento y el país de procedencia de los jugadores. Para cada estimación se obtuvo el set de variables que mejoró la capacidad predictiva.

Predicción de abandonos

En la Tabla I se muestran los resultados de las estimaciones de los modelos, para cada período de observación y de abandono. Se resumen los modelos que mejor AUC presentaron luego de estimar distintas variantes, analizar el grado de significatividad estadística de las variables y comprobar que los resultados estaban en consonancia con los conocimientos empíricos y teóricos sobre el comportamiento de jugadores de videojuegos (signos esperados). En todos los casos, los modelos estimados fueron en su conjunto estadísticamente significativos, al lograrse un p-valor de *Prob > chi2* muy cercano a cero.

Tabla I. Resultados de la regresión logística por cada período. Variable dependiente *churn*.

	(1) PO=1, PA=3	(2) NORM PO=1, PA=3	(3) PO=7, PA=7	(4) NORM PO=7, PA=7	(5) PO=14, PA=16	(6) NORM PO=14, PA=16
VARIABLES	**churn**	**churn**	**churn**	**churn**	**churn**	**churn**
revenue					0.0001633**	0.0299220**
					(0.0000720)	(0.0132009)
sessions	-0.2648143***	-0.9896164***	-0.0863161***	-0.8460261***	-0.0488157***	-0.6739280***
	(0.0066609)	(0.0248918)	(0.0028265)	(0.0277042)	(0.0020089)	(0.0277344)
missions_started	0.1664424***	3.6443550***	0.0142396***	0.5334771***	0.0023761*	0.1111236*
	(0.0100988)	(0.2211188)	(0.0030337)	(0.1136568)	(0.0013546)	(0.0633521)

missions_completed	-0.2153308***	-3.3844051***	-0.0399313***	-0.8551291***	-0.0215566***	-0.5179567***
	(0.0099878)	(0.1569803)	(0.0030915)	(0.0662055)	(0.0015640)	(0.0375783)
missions_aborted	-0.1171256***	-0.2451092***	-0.0296554***	-0.1471834***	-0.0246847***	-0.1644246***
	(0.0113749)	(0.0238043)	(0.0045146)	(0.0224066)	(0.0032626)	(0.0217321)
missions_failed	-0.1458099***	-1.2001657***	-0.0065778**	-0.1281836**	0.0025701***	0.0720176***
	(0.0103505)	(0.0851953)	(0.0030373)	(0.0591894)	(0.0009937)	(0.0278444)
total_time_ms						
transactions					-0.0768015**	-0.0313566**
					(0.0354327)	(0.0144665)
2.tier	0.0982340***	0.0982340***	0.2620265***	0.2620265***	0.2342800***	0.2342800***
	(0.0331106)	(0.0331106)	(0.0363772)	(0.0363772)	(0.0377736)	(0.0377736)
3.tier	0.1802874***	0.1802874***	0.4257710***	0.4257710***	0.5217856***	0.5217856***
	(0.0233714)	(0.0233714)	(0.0261648)	(0.0261648)	(0.0279215)	(0.0279215)
Constante	1.8605506***	0.3449392***	2.1927987***	1.1460376***	2.2485051***	1.4386785***
	(0.0210292)	(0.0154339)	(0.0216647)	(0.0164362)	(0.0222020)	(0.0171264)
Observaciones entrenamiento	53,336	53,336	53,471	53,471	53,516	53,516
Prob > chi	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
AUC	**0.8289**	**0.8289**	**0.8303**	**0.8303**	**0.7948**	**0.7948**
Precisión positivos	76.18%	76.18%	83.09%	83.09%	84.35%	84.35%
Precisión negativos	76.31%	76.31%	69.16%	69.16%	64.65%	64.65%

NORM.: Datos normalizados

Errores estándares entre paréntesis
*** p<0.01, ** p<0.05, * p<0.1

El comando logit de Stata devuelve los coeficientes de la regresión en unidades de *log-odds*, por lo que la ecuación de la regresión logística del modelo 1 (PO=1 y PA=3), sería:

$$\text{Ln}\left(\frac{P}{1-P}\right) = -0.2648 * sessions + 0.1664 * missions\_started - 0.2153 * missions\_completed - 0.1171 * missions\_aborted - 0.1458 * missions\_failed + 0.0982 * tier2 + 0.1803 * tier3 + 1.8606$$

El aumento de la probabilidad de abandonos depende de las variables con coeficiente positivo, y viceversa. Lo mismo aplica para las *dummy de* nivel de país (*tier*), aunque sus coeficientes se interpretan respecto a la categoría de referencia (*tier1*). La interpretación de la constante es el valor esperado de la *log-odds* de *churn*, cuando todos los predictores son iguales a cero.

A modo de ejemplo, en el modelo 1 el aumento de sesiones jugadas, misiones completadas, misiones abortadas y misiones fallidas, disminuye la probabilidad de que los usuarios realicen abandonos. En caso contrario, mientras más misiones iniciadas tenga el usuario, mayor es la probabilidad de que abandone el juego. El hecho de ser usuarios de países *tier2* o *tier3*, aumenta la probabilidad de abandono respecto a los usuarios de países *tier1*. Para ejemplificar la interpretación del valor del coeficiente de misiones iniciadas en el modelo 1, se puede indicar

que: por cada unidad que este aumente, se espera un aumento de 0.1664 en la *log-odds* de *churn* (abandonos), manteniendo constantes las demás variables independientes. El valor de la constante del modelo (1.8606) indica una probabilidad de abandono P=0.8654 cuando todos los demás predictores están en cero.

Comparando los modelos en los distintos períodos de observación y abandono, se aprecia que las variables relacionadas con pagos (*revenue* y *transactions*) solo son estadísticamente significativas para predecir abandonos en períodos a largo plazo. Es decir, el hecho de que los usuarios paguen en períodos a corto/medio plazo no influye en su decisión de continuar jugando en esos mismos plazos. En cuanto a misiones fallidas, se observa un cambio de signo en el largo plazo, respecto al corto y medio. Esto significa que el hecho de perder más misiones no aumenta la probabilidad de abandono en los primeros días de juego, pero mientras más se juega, las derrotas comienzan a influir positivamente en la decisión de abandonar. El comportamiento respecto al país de los usuarios es el mismo para todos los períodos (ver comentario sobre países en el párrafo anterior). Por su parte, el tiempo destinado a jugar no resultó estadísticamente significativo para ninguno de los modelos.

Respecto al poder predictivo, se pudo obtener mejor AUC en el período de medio plazo, siendo el de largo plazo el de menor capacidad predictiva. La precisión de resultados positivos (abandonos clasificados correctamente), incrementó conforme aumentaron los períodos de observación y abandono. Lo contrario ocurrió con la precisión de negativos (no abandonos): a medida que aumentaron los plazos de observación y abandono, peor los clasificó el método de regresión logística.

Los resultados de las clasificaciones de abandono mediante regresión logística fueron los mismos para datos normalizados y no normalizados.

Predicción de pagos

Para la predicción de los pagos se procedió de forma similar a la predicción de abandonos, en cuanto a la formulación de los modelos y la elección de las mejores estimaciones. Cabe destacar que a diferencia de los modelos para predecir abandonos, en la predicción de pagos las variables *revenue*, *transactions* y *total_time_ms* ocasionaron problemas de convergencia en algunos sets. Debido a la "rareza" de la clase pagadores, se estimaron los modelos sobre los datos originales y

también sobre los datos con sobre-muestreo de pagadores (la estrategia y proporción del sobre-muestreo fue explicada en el apartado de Transformación). En la siguiente tabla se resumen los resultados de las estimaciones sobre los datos originales, para cada período de observación y de pago:

Tabla II. Resultados de la regresión logística por cada período. Variable dependiente *payer* (sin sobre-muestreo).

	(7) PO=1, PP=3	(8) NORM PO=1, PP=3	(9) PO=7, PP=7	(10) NORM PO=7, PP=7	(11) PO=14, PP=16	(12) NORM PO=14, PP=16
VARIABLES	**payer**	**payer**	**payer**	**payer**	**payer**	**payer**
revenue	-0.0005903**	-0.0743417**				
	(0.0002713)	(0.0341680)				
sessions	0.0561934***	0.2098200***	0.0347526***	0.3400646***	0.0141649***	0.1960242***
	(0.0154283)	(0.0576077)	(0.0060243)	(0.0589491)	(0.0047026)	(0.0650777)
missions_started			0.0041914*	0.1570193*	0.0031663*	0.1474736*
			(0.0024096)	(0.0902707)	(0.0018768)	(0.0874159)
missions_completed	0.0292078***	0.4601291***	0.0087353**	0.1868831**	0.0093494***	0.2252138***
	(0.0037234)	(0.0586571)	(0.0040627)	(0.0869175)	(0.0031163)	(0.0750662)
missions_aborted						
missions_failed						
total_time_ms						
transactions	1.3543086***	0.2121831***	0.6893613***	0.2372216***	0.5701732***	0.2389411***
	(0.1346274)	(0.0210924)	(0.0497606)	(0.0171235)	(0.0433493)	(0.0181663)
2.tier	-0.4681310*	-0.4681310*	-1.7803751***	-1.7803751***	-1.6017868***	-1.6017868***
	(0.2433114)	(0.2433114)	(0.4795673)	(0.4795673)	(0.4958920)	(0.4958920)
3.tier	-1.2182986***	-1.2182986***	-1.0974054***	-1.0974054***	-0.6843899***	-0.6843899***
	(0.2266679)	(0.2266679)	(0.2537365)	(0.2537365)	(0.2369728)	(0.2369728)
Constante	-6.2843268***	-5.6463016***	-6.6206222***	-6.0803674***	-6.5994272***	-6.1651872***
	(0.1378245)	(0.1093345)	(0.1570911)	(0.1328853)	(0.1538867)	(0.1369260)
Observaciones entrenamiento	46,669	46,669	46,787	46,787	46,826	46,826
Prob > chi	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
AUC	**0.8769**	**0.8769**	**0.8903**	**0.8903**	**0.8878**	**0.8878**
Precisión positivos	33.33%	33.33%	57.14%	57.14%	36.36%	36.36%
Precisión negativos	99.62%	99.62%	99.78%	99.78%	99.72%	99.72%

NORM.: Datos normalizados

Errores estándares entre paréntesis
*** p<0.01, ** p<0.05, * p<0.1

En la tabla anterior se puede observar como los usuarios de países *tier2* y *tier3* son menos propensos a realizar pagos que los jugadores de países *tier1*. Un usuario que realiza una compra o IAP (información almacenada en la variable *transactions*), es más atraído a volver a realizar algún pago en el futuro. Sin embargo, mientras mayor cantidad de dinero (*revenue*) aporta un usuario el primer día, menor es la probabilidad de que este vuelva a pagar en los siguientes tres

días. La cantidad de misiones iniciadas tiene efecto positivo en la decisión de pagar en el medio plazo y largo plazo, pero no es estadísticamente significativa en el corto plazo. Algunas variables no formaron parte de los modelos en ningún período: misiones abortadas, misiones fallidas y tiempo total de juego.

A continuación se muestra el resumen de los resultados de las estimaciones, con sobre-muestreo de la clase pagadores:

Tabla III. Resultados de la regresión logística por cada período. Variable dependiente *payer* (con sobre-muestreo).

	(13) PO=1, PP=3	(14) NORM PO=1, PP=3	(15) PO=7, PP=7	(16) NORM PO=7, PP=7	(17) PO=14, PP=16	(18) NORM PO=14, PP=16
VARIABLES	**payer**	**payer**	**payer**	**payer**	**payer**	**payer**
revenue						
sessions	0.0402090***	0.1836663***	0.0207480***	0.3310213***	0.0060372***	0.1441353***
	(0.0035798)	(0.0163518)	(0.0020758)	(0.0331180)	(0.0014664)	(0.0350085)
missions_started			0.0226094***	1.3987418***	0.0316191***	2.6537024***
			(0.0025619)	(0.1584935)	(0.0017412)	(0.1461360)
missions_completed	0.0544324***	1.0255623***	0.0172926***	0.5124518***	0.0041375**	0.1451368**
	(0.0010475)	(0.0197365)	(0.0024786)	(0.0734499)	(0.0018080)	(0.0634203)
missions_aborted	0.0194620***	0.0475761***	-0.0247334***	-0.1791493***	-0.0359402***	-0.3768220***
	(0.0066431)	(0.0162394)	(0.0029916)	(0.0216688)	(0.0024067)	(0.0252331)
missions_failed	0.0419905***	0.4674260***	0.0042323*	0.1624321*	-0.0109176***	-0.5925578***
	(0.0017742)	(0.0197502)	(0.0024864)	(0.0954283)	(0.0017125)	(0.0929471)
total_time_ms	-0.0000000***	-0.3121655***	-0.0000000***	-0.2414720***		
	(0.0000000)	(0.0221342)	(0.0000000)	(0.0306565)		
transactions						
2.tier	-0.6133396***	-0.6133396***	-1.7659166***	-1.7659166***	-2.1488202***	-2.1488202***
	(0.0340435)	(0.0340435)	(0.0549099)	(0.0549099)	(0.0603006)	(0.0603006)
3.tier	-1.2944985***	-1.2944984***	-1.3562634***	-1.3562634***	-0.8997625***	-0.8997625***
	(0.0288942)	(0.0288942)	(0.0319551)	(0.0319551)	(0.0285043)	(0.0285043)
Constante	-2.3445557***	-0.9911661***	-2.7374422***	-0.9945718***	-2.7032944***	-0.9897305***
	(0.0221125)	(0.0140933)	(0.0251450)	(0.0153942)	(0.0236018)	(0.0156519)
Observaciones entrenamiento	61,819	61,819	62,058	62,058	62,138	62,138
Prob > chi	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
AUC	**0.8667**	**0.8667**	**0.8824**	**0.8824**	**0.8710**	**0.8710**
Precisión positivos	2.65%	2.65%	1.75%	1.75%	2.09%	2.09%
Precisión negativos	99.80%	99.80%	99.86%	99.86%	99.80%	99.80%

NORM.: Datos normalizados

Errores estándares entre paréntesis
*** p<0.01, ** p<0.05, * p<0.1

Como se puede apreciar en la Tabla III, el sobre-muestreo de pagadores provocó un gran descenso en la precisión de la clase positiva. Es decir, un gran número de usuarios clasificados

como pagadores, no pagaron en realidad. Por este motivo, se concluye que la regresión logística funcionó mejor para la muestra original.

En cuanto al poder predictivo de la muestra original (Tabla II), se obtuvo mejor AUC en el período de medio plazo (al igual que sucedió en la predicción de abandonos), siendo el período de corto plazo el de menor capacidad predictiva. Se puede apreciar que el método de Regresión Logística presentó problemas de predicción en la clase minoritaria: la mejor precisión de positivos fue de 57.14% (modelos 9 y 10 en el medio plazo), y la peor fue de 33.33% (modelos 7 y 8 en el corto plazo).

Los resultados de las clasificaciones de pagadores mediante regresión logística fueron los mismos para datos normalizados y no normalizados.

6.2 Bosques Aleatorios (Random Forests)

Los algoritmos de *Random Forests* fueron ejecutados en Python, utilizando distintos métodos del módulo *sklearn*.

Para obtener los hiperparámetros óptimos se utilizó el método *GridSearchCV*, el cual permite predefinir rangos de hiperparámetros para luego evaluar todas las combinaciones posibles en el clasificador, mediante validación cruzada *K-Fold*. De esta forma, se realizaron varios ajustes sobre los datos de entrenamiento, a través del método *RandomForestClassifier*. Los valores de hiperparámetros que se probaron fueron:

- *n_estimators*: Número de árboles en el bosque: 100, 120 y 150. El valor por defecto del método *RandomForestClassifier* es 100.

- *max_features*: Número de variables a considerar al buscar la mejor división: ‘sqrt’ (raíz cuadrada del número de variables) y ‘log2’ (logaritmo en base dos del número de variables). El valor por defecto es ‘auto’, que hace el mismo cálculo que ‘sqrt’.

- *min_samples_split*: Número mínimo de muestras necesarias para dividir un nodo interno: 2, 5 y 10. Valor por defecto: 2.

- *min_samples_leaf*: Número mínimo de muestras requeridas para estar en un nodo hoja: 1, 2 y 4. Un punto de división a cualquier profundidad solo se considerará si deja al menos

min_samples_leaf muestras de entrenamiento en cada una de las ramas izquierda y derecha. Esto puede tener el efecto de suavizar el modelo. Valor por defecto: 1.

- *bootstrap*: Utilizar, o no, muestras *bootstrap* al construir los árboles: verdadero y falso. Valor por defecto: verdadero.

Los hiperparámetros anteriores derivaron en 108 combinaciones posibles. Para los ajustes se empleó *K-Fold* con *K=3*, por lo que se procesaron un total de 108*3=324 ajustes en cada set de validación cruzada. Se conformaron varios sets de validaciones cruzadas, cada uno con una medida para determinar la mejor combinación de hiperparámetros:

- *mean_accuracy*: Precisión media de las clasificaciones. Utilizada en abandonos y pagos.
- *roc_auc*: Área bajo la curva ROC. Utilizada en abandonos y pagos.
- *balanced_accuracy*: Precisión equilibrada para tratar con conjuntos de datos desbalanceados en problemas de clasificación binaria. Utilizada en pagos.

Además, se procesó una última clasificación con todos los parámetros por defecto que provee *RandomForestClassifier*. Finalmente se pudieron contrastar un total de 649 modelos para predecir abandonos y 973 modelos para predecir pagos, por cada conjunto de datos.

Para los ajustes se utilizaron 6 núcleos paralelos de procesamiento, en un ordenador portátil con procesador Intel Core i7-4710HQ a 2.5 GHz.

Predicción de abandonos

En la siguiente tabla se muestran los resultados de las predicciones de abandonos para los distintos períodos de observación (PO) y abandono (PA). Se presentan los resultados de los modelos con datos normalizados y sin normalizar, para contrastar cómo la estandarización de los datos puede influir en la precisión de los clasificadores. Además, se muestra el AUC que obtuvo cada modelo en los datos de validación, para los hiperparámetros por defecto que establece Python, y para los hiperparámetros afinados. La tupla “Puntuación validación cruzada” contiene información sobre el criterio de evaluación utilizado en la iteración con los mejores hiperparámetros. Se muestra además información sobre el tiempo que tardó la validación cruzada

con mejores resultados en el entrenamiento, así como la precisión de clasificaciones positivas/negativas en el set de prueba.

Tabla IV. Resultados de *Random Forest* por cada período. Variable a predecir: *churn*.

Hiperparámetros óptimos	(19) PO=1, PA=3	(20) NORM PO=1, PA=3	(21) PO=7, PA=7	(22) NORM PO=7, PA=7	(23) PO=14, PA=16	(24) NORM PO=14, PA=16
n_estimators	120	120	150	100	150	150
max_features	sqrt	sqrt	sqrt	sqrt	sqrt	sqrt
min_samples_split	10	10	10	10	10	10
min_samples_leaf	4	4	4	4	4	4
bootstrap	True	True	True	True	True	True
Observaciones entrenamiento	53,336	53,336	53,471	53,471	53,516	53,516
Tiempo validación cruzada	8.3 min	8.3 min	8.5 min	8.3 min	8.5 min	8.4 min
Puntuación validación cruzada	roc_auc	roc_auc	roc_auc	mean_accuracy	roc_auc	mean_accuracy
AUC hiperparámetros por defecto	0.7788	0.7787	0.7926	0.7912	0.7572	0.7590
AUC hiperparámetros afinados	**0.8291**	**0.8291**	**0.8386**	**0.8380**	**0.8092**	**0.8094**
Precisión positivos	80.08%	80.09%	84.72%	84.69%	85.11%	85.14%
Precisión negativos	71.67%	71.69%	64.69%	64.43%	62.07%	61.74%

NORM: Datos normalizados

Como se observa en la tabla, los hiperparámetros óptimos fueron similares en todos los conjuntos de datos, variando únicamente el número de árboles en el bosque (*n_estimators*). El uso de datos normalizados trasformó en pequeña medida los resultados de las clasificaciones, favoreciendo o desfavoreciendo el AUC y la precisión, pero sin llegar a tener un efecto relevante. Los hiperparámetros afinados mejoraron el valor del AUC en todos los casos, en comparación con los hiperparámetros por defecto. El período en el cual se clasificaron mejor los datos según el AUC fue el de medio plazo, seguido por el corto plazo. Como se observa en la precisión, *Random Forest* clasificó con mayor exactitud a la clase mayoritaria (positivos o abandonos).

Predicción de pagos

El bajo porciento de pagadores provocó que las clasificaciones de pagos fuesen menos precisas que las de abandonos. En las siguientes tablas se muestran los resultados de las predicciones de pagadores para los distintos períodos de observación (PO) y pago (PP), con datos normalizados y sin normalizar, sobre la muestra original y con sobre-muestreo. Se provee la misma información que se comentó anteriormente en la predicción de abandonos.

Tabla V. Resultados de *Random Forest* por cada período, sin sobre-muestreo. Variable a predecir: *payer*.

Hiperparámetros óptimos	(25) PO=1, PP=3	(26) NORM PO=1, PP=3	(27) PO=7, PP=7	(28) NORM PO=7, PP=7	(29) PO=14, PP=16	(30) NORM PO=14, PP=16
n_estimators	150	150	100	100	150	150
max_features	sqrt	sqrt	sqrt	sqrt	sqrt	sqrt
min_samples_split	2	2	5	5	2	2
min_samples_leaf	4	4	1	1	4	4
bootstrap	True	True	False	False	True	True
Observaciones entrenamiento	46,669	46,669	46,787	46,787	46,826	46,826
Tiempo validación cruzada	4.4 min	4.4 min	4.6 min	4.6 min	4.4 min	4.3 min
Puntuación validación cruzada	roc_auc	roc_auc	balanced_accuracy	balanced_accuracy	mean_accuracy	mean_accuracy
AUC hiperparámetros por defecto	0.6925	0.6924	0.7118	0.7116	0.6861	0.6861
AUC hiperparámetros afinados	**0.8397**	**0.8398**	**0.7983**	**0.7982**	**0.7980**	**0.7979**
Precisión positivos	**20.00%**	**20.00%**	**20.00%**	**20.00%**	**25.00%**	**25.00%**
Precisión negativos	99.62%	99.62%	99.78%	99.78%	99.71%	99.71%

NORM: Datos normalizados

Tabla VI. Resultados de *Random Forest* por cada período, con sobre-muestreo. Variable a predecir: *payer*.

Hiperparámetros óptimos	(31) PO=1, PP=3	(32) NORM PO=1, PP=3	(33) PO=7, PP=7	(34) NORM PO=7, PP=7	(35) PO=14, PP=16	(36) NORM PO=14, PP=16
n_estimators	100	100	100	100	100	100
max_features	sqrt	sqrt	sqrt	sqrt	sqrt	sqrt
min_samples_split	2	2	2	2	2	2
min_samples_leaf	1	1	1	1	1	1
bootstrap	True	True	True	True	False	False
Observaciones entrenamiento	61,819	61,819	62,058	62,058	62,138	62,138
Tiempo validación cruzada	6.1 min	6.2 min	5.9 min	5.8 min	5.8 min	5.6 min
Puntuación validación cruzada	roc_auc	roc_auc	roc_auc	roc_auc	balanced_accuracy	balanced_accuracy
AUC hiperparámetros por defecto	0.7161	0.7160	0.7427	0.7427	0.7018	0.7017
AUC hiperparámetros afinados	**0.7576**	**0.7577**	**0.8012**	**0.8012**	**0.7619**	**0.7619**
Precisión positivos	**46.15%**	**46.15%**	**16.67%**	**16.37%**	**28.57%**	**30.77%**
Precisión negativos	99.67%	99.67%	99.78%	99.78%	99.72%	99.72%

NORM: Datos normalizados

Al analizar ambos resultados se observa un comportamiento ya no tan generalizable como en el caso de los abandonos; los hiperparámetros óptimos fueron variando en dependencia del conjunto de datos objetivo. El uso de datos normalizados no tuvo un efecto relevante sobre las clasificaciones, aunque las modificó en pequeña medida. Por otra parte, el AUC de todos los modelos mejoró respecto al AUC del modelo con parámetros por defecto. Los conjuntos con sobre-muestreo mejoraron el AUC en el medio plazo, pero lo empeoraron en el corto y largo plazo. *Random Forest* clasificó con mayor exactitud a la clase mayoritaria (negativos o no

pagadores), siendo los pagadores a corto plazo los que mejor fue capaz de clasificar en la clase minoritaria (46.15%, modelos 31 y 32).

6.3 Potenciación del Gradiente (Gradient Boosting)

Los algoritmos de *Gradient Boosting* fueron ejecutados en Python, utilizando distintos métodos del módulo *sklearn*.

De forma similar a *Random Forest*, se obtuvieron los hiperparámetros óptimos sobre el set de entrenamiento mediante validación cruzada, a través del método *GridSearchCV*. De este modo, se realizaron varios ajustes invocando al método *GradientBoostingClassifier*. Los valores de hiperparámetros que se probaron fueron:

- *n_estimators*: Número de etapas *boosting* a realizar: 100, 120 y 150. El valor por defecto del método es 100.

- *max_features*: Número de variables a considerar al buscar la mejor división: 'sqrt' (raíz cuadrada del número de variables) y *n_features* (cantidad de variables del conjunto). El valor por defecto es *n_features*.

- *min_samples_split*: Número mínimo de muestras necesarias para dividir un nodo interno: 2, 5 y 10. Valor por defecto: 2.

- *min_samples_leaf*: Número mínimo de muestras requeridas para estar en un nodo hoja: 1, 2 y 4. Un punto de división a cualquier profundidad solo se considerará si deja al menos *min_samples_leaf* muestras de entrenamiento en cada una de las ramas izquierda y derecha. Esto puede tener el efecto de suavizar el modelo. Valor por defecto: 1.

- *loss*: Función de pérdida a optimizar: 'deviance' que se refiere a la desviación (regresión logística) para la clasificación con resultados probabilísticos y 'exponential' (función de pérdida exponencial). Valor por defecto: 'deviance'.

Los hiperparámetros anteriores derivaron en 108 combinaciones posibles. Los ajustes se realizaron de forma similar a *Random Forest*, empleando validación cruzada *K-Fold* con *K=3*, procesándose 324 ajustes en cada set de validación cruzada. Se conformaron varios sets para

validación cruzada, cada uno con una medida para determinar la mejor combinación de hiperparámetros (mismos criterios que en *Random Forest*).

Además, se procesó una clasificación con todos los parámetros por defecto de *GradientBoostingClassifier*. Finalmente se pudieron contrastar un total de 649 modelos para predecir abandonos y 973 modelos para predecir pagos, por cada conjunto de datos.

Se utilizó la misma capacidad de procesamiento computacional que para *Random Forest*.

Predicción de abandonos

En la siguiente tabla se muestran los resultados de las predicciones de abandonos para los distintos períodos de observación (PO) y abandono (PA).

Tabla VII. Resultados de *Gradient Boosting* por cada período. Variable a predecir: *churn*.

Hiperparámetros óptimos	(37) PO=1, PP=3	(38) NORM PO=1, PP=3	(39) PO=7, PP=7	(40) NORM PO=7, PP=7	(41) PO=14, PP=16	(42) NORM PO=14, PP=16
n_estimators	150	150	150	120	150	100
max_features	n_features	n_features	n_features	n_features	sqrt	n_features
min_samples_split	2	2	10	10	10	10
min_samples_leaf	4	4	1	4	2	4
loss	exponential	exponential	exponential	deviance	exponential	deviance
Observaciones entrenamiento	53,336	53,336	53,471	53,471	53,516	53,516
Tiempo validación cruzada	6.5 min	6.4 min	6.5 min	6.4 min	6.6 min	6.5 min
Puntuación validación cruzada	roc_auc	roc_auc	roc_auc	mean_accuracy	roc_auc	mean_accuracy
AUC hiperparámetros por defecto	0.8329	0.8329	0.8445	0.8445	0.8204	0.8204
AUC hiperparámetros afinados	**0.8337**	**0.8337**	**0.8450**	**0.8446**	**0.8204**	**0.8208**
Precisión positivos	79.71%	79.71%	84.98%	84.99%	84.91%	84.86%
Precisión negativos	72.19%	72.19%	65.89%	66.28%	65.00%	63.89%

NORM: Datos normalizados

Como se observa en la tabla, los hiperparámetros óptimos resultaron diferentes por cada conjunto de datos. El resto de patrones coincidieron con los detectados en la predicción de abandonos de *Random Forest*: los datos normalizados transformaron en pequeña medida los resultados de las clasificaciones, pero de forma irrelevante; los hiperparámetros afinados mejoraron el valor del AUC en todos los casos, en comparación con los hiperparámetros por defecto; el período en el cual se clasificaron mejor los datos según el AUC fue el de medio plazo, seguido por el corto plazo. *Gradient Boosting* clasificó con mayor exactitud a la clase mayoritaria (positivos o abandonos).

Predicción de pagos

Una vez más la presencia de datos desbalanceados provocó clasificaciones de pagos menos precisas que las clasificaciones de abandonos. En las siguientes tablas se muestran los resultados de las predicciones de pagadores para los distintos períodos de observación (PO) y pago (PP), con datos normalizados y sin normalizar, sobre la muestra original y con sobre-muestreo.

Tabla VIII. Resultados de *Gradient Boosting* por cada período. Variable a predecir: *payer* (sin sobre-muestreo)

Hiperparámetros óptimos	**(43) PO=1, PP=3**	**(44) NORM PO=1, PP=3**	**(45) PO=7, PP=7**	**(46) NORM PO=7, PP=7**	**(47) PO=14, PP=16**	**(48) NORM PO=14, PP=16**
n_estimators	Por defecto	Por defecto	100	100	100	100
max_features	Por defecto	Por defecto	n_features	sqrt	sqrt	sqrt
min_samples_split	Por defecto	Por defecto	5	5	2	2
min_samples_leaf	Por defecto	Por defecto	1	2	4	4
loss	Por defecto	Por defecto	deviance	exponential	exponential	exponential
Observaciones entrenamiento	46,669	46,669	46,787	46,787	46,826	46,826
Tiempo validación cruzada	5.1 min	4.8 min	5.1 min	5.1 min	5.1 min	5.2 min
Puntuación validación cruzada	-	-	balanced_accuracy	roc_auc	roc_auc	roc_auc
AUC hiperparámetros por defecto	**0.8764**	**0.8764**	0.6880	0.6880	0.7979	0.7979
AUC hiperparámetros afinados	0.8772	0.8772	**0.7005**	**0.9081**	**0.9071**	**0.9071**
Precisión positivos	**34.38%**	**34.38%**	**23.81%**	**50.00%**	**50.00%**	**50.00%**
Precisión negativos	99.67%	99.67%	99.79%	99.77%	99.71%	99.71%

NORM: Datos normalizados

Tabla IX. Resultados de *Gradient Boosting* por cada período. Variable a predecir: *payer* (con sobre-muestreo).

Hiperparámetros óptimos	**(49) PO=1, PP=3**	**(50) NORM PO=1, PP=3**	**(51) PO=7, PP=7**	**(52) NORM PO=7, PP=7**	**(53) PO=14, PP=16**	**(54) NORM PO=14, PP=16**
n_estimators	150	150	150	150	150	150
max_features	n_features	n_features	n_features	n_features	n_features	n_features
min_samples_split	2	2	2	2	2	2
min_samples_leaf	1	1	4	4	4	4
loss	deviance	deviance	deviance	deviance	deviance	deviance
Observaciones entrenamiento	61,819	61,819	62,058	62,058	62,138	62,138
Tiempo validación cruzada	7.7 min	7.4 min	7.9 min	7.3 min	7.9 min	8.7 min
Puntuación validación cruzada	roc_auc	roc_auc	roc_auc	roc_auc	roc_auc	roc_auc
AUC hiperparámetros por defecto	0.8628	0.8628	0.9039	0.9039	0.8912	0.8912
AUC hiperparámetros afinados	**0.8560**	**0.8560**	**0.9066**	**0.9066**	**0.8775**	**0.8775**
Precisión positivos	**5.66%**	**5.66%**	**7.12%**	**7.12%**	**10.43%**	**10.48%**
Precisión negativos	99.78%	99.78%	99.87%	99.87%	99.81%	99.81%

NORM: Datos normalizados

Al analizar las tablas se observa que los hiperparámetros óptimos fueron variando en dependencia del conjunto de datos analizado, existiendo mayor homogeneidad en los datos sobre-muestreados. El uso de datos normalizados tuvo un efecto relevante en el AUC y la precisión de positivos en el set original de medio plazo (modelos 45 y 46), mejorando los

resultados considerablemente: el AUC incrementó de 0.7005 a 0.9081 y la precisión de positivos pasó de 23.81% a 50.00%. En el resto de los modelos no se notaron cambios relevantes al emplear datos normalizados. Por otra parte, el AUC de los modelos con mejores hiperparámetros no fue en todos los casos superior al AUC del modelo por defecto. En algunos casos se seleccionó como modelo ganador aquel que, si bien no tuvo mayor AUC, presentó una precisión de positivos mayor a cero (modelos 49, 50, 53 y 54). En el set original de corto plazo (modelos 43 y 44), el modelo por defecto presentó mejores resultados en los datos de validación que los afinados. *Gradient Boosting* clasificó con mayor exactitud a la clase mayoritaria (negativos o no pagadores), siendo los pagadores a medio y largo plazo los que mejor fue capaz de clasificar en la clase minoritaria (50.00%, modelos 46, 47 y 48).

6.4 Comparando los modelos

El análisis de los resultados de los métodos de Regresión Logística, Bosques Aleatorios y Potenciación del Gradiente, evidencia la variabilidad con que cada algoritmo realizó las clasificaciones. En el presente epígrafe se comparan los mejores modelos de clasificación para cada variable dependiente. Se realiza un resumen por cada período de observación y abandono/pago, ya que como se comentó en los apartados anteriores, no se observaron patrones genéricos para todos los períodos. El criterio para definir a un modelo como el mejor dentro de su tipo en un mismo período fue el AUC.

Una vez determinados por cada período los mejores candidatos de cada método de clasificación, se puede decidir de forma más sencilla qué método utilizar para la predicción, siendo consecuentes con las necesidades de la empresa (priorizar AUC o priorizar precisión). El resumen se presenta en las siguientes tablas:

Tabla X. Comparación de mejores modelos por período para predecir abandonos.

		Criterios de evaluación			
Períodos de Observación / Abandono	**Modelo / No.**	**AUC**	**Precisión positivos**	**Precisión negativos**	**Normalizado**
P.O=1, P.A=3	Logit #1	0.8289	76.18%	**76.31%**	X
	R.F #20	0.8291	**80.09%**	71.69%	
	G.B #37	**0.8337**	79.71%	72.19%	
P.O=7, P.A=7	Logit #3	0.8303	83.09%	**69.16%**	
	R.F #21	0.8386	84.72%	64.69%	
	G.B #39	**0.8450**	**84.98%**	65.89%	
	Logit #5	0.7948	84.35%	**64.65%**	

P.O=14, P.A=16	R.F #24	0.8094	**85.14%**	61.74%	X
	G.B #42	**0.8208**	84.86%	63.89%	X

Tabla XI. Comparación de mejores modelos por período para predecir pagos.

		Criterios de evaluación				
Períodos de Observación / Pago	**Modelo / No.**	**AUC**	**Precisión positivos**	**Precisión negativos**	**Normalizado**	**Sobre-muestreo**
P.O=1, P.P=3	Logit #7	**0.8769**	33.33%	99.62%		
	R.F #26	0.8398	20.00%	99.62%	**X**	
	G.B #43	0.8764	**34.38%**	**99.67%**		
P.O=7, P.P=7	Logit #9	0.8903	**57.14%**	**99.78%**		
	R.F #33	0.8012	16.67%	**99.78%**		**X**
	G.B #46	**0.9081**	50.00%	99.77%	**X**	
P.O=14, P.P=16	Logit #11	0.8878	36.36%	99.72%		
	R.F #29	0.7980	25.00%	99.71%		
	G.B #47	**0.9071**	**50.00%**	**99.74%**		

De manera general, los métodos de clasificación binaria lograron buenos valores de AUC. Sin embargo, es importante señalar que en el caso de los pagos, la precisión de positivos alcanzada no fue óptima: los métodos clasificaron como pagadores a varios usuarios que en realidad no realizaron pagos. Esto se debió al bajo porcentaje de usuarios pagadores (como se explicó anteriormente, menos del 0.5% del total de la muestra). Dicha situación se intentó revertir con una técnica de sobre-muestreo, que solo fue efectiva en algunos de los modelos. El uso de datos normalizados mejoró los resultados en algunos casos, como se aprecia en las tablas anteriores. El método que en general mejor AUC obtuvo fue *Gradient Boosting*.

Una de las principales ventajas de usar bosques aleatorios en las tareas de clasificación es poder conocer la importancia de las variables. En *Random Forest*, para cada característica, los valores de importancia indican la cantidad de heterogeneidad que se perdería al no usar esa variable en la clasificación (Sifa, et al., 2015).

La importancia relativa de las variables fue procesada en Python, para los métodos *Random Forest* y *Gradient Boosting*. Las figuras II, III y IV muestran estos valores para las predicciones de abandonos a corto, medio y largo plazo, en los modelos seleccionados como los mejores.

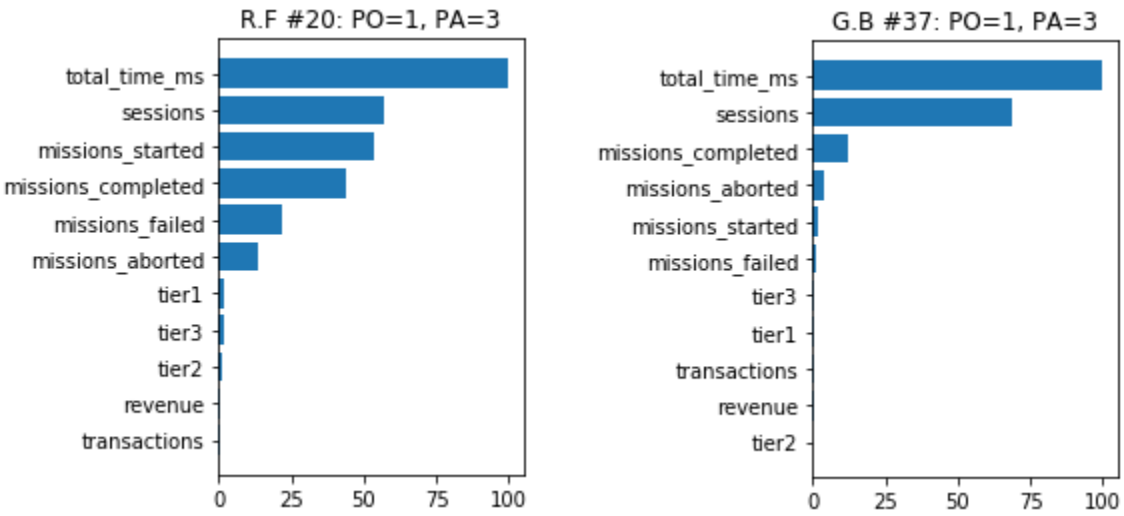

Figura II. Importancia relativa de las variables: abandonos a corto plazo. Izquierda: *Random Forest*, derecha: *Gradient Boosting*. Fuente: elaboración propia.

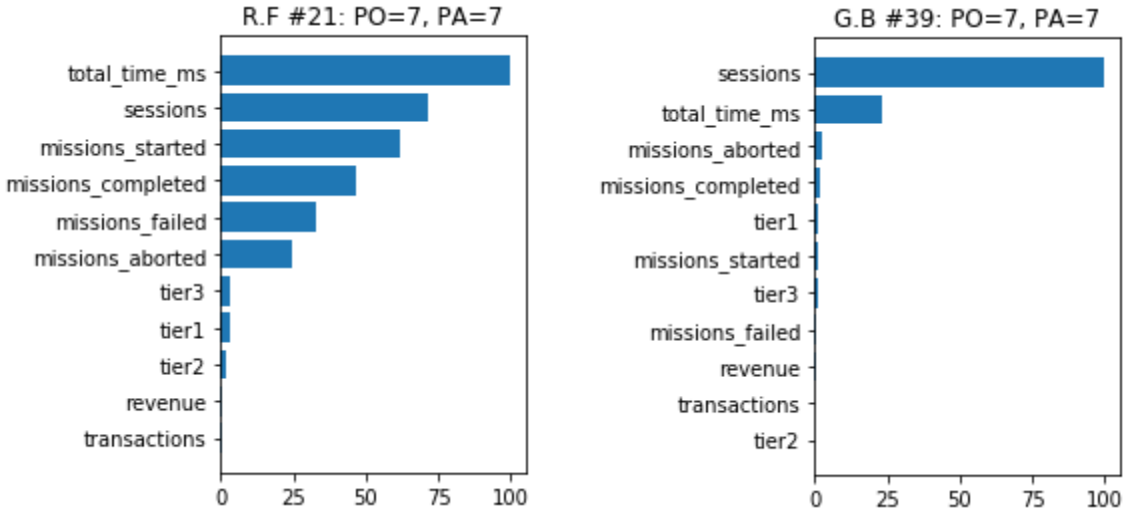

Figura III. Importancia relativa de las variables: abandonos a medio plazo. Izquierda: *Random Forest*, derecha: *Gradient Boosting*. Fuente: elaboración propia.

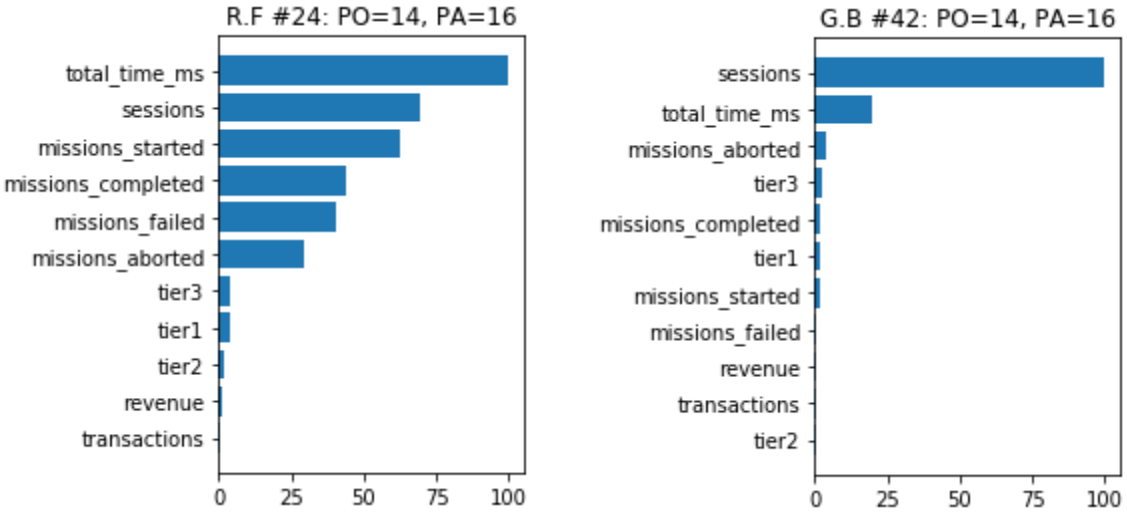

Figura IV. Importancia relativa de las variables: abandonos a largo plazo. Izquierda: *Random Forest*, derecha: *Gradient Boosting*. Fuente: elaboración propia.

Como se observa en las figuras anteriores, *Gradient Boosting* se centró en la información contenida en las variables *total_time_ms* y *sessions* para predecir los abandonos, mientras que *Random Forest* se nutrió de información de esos y otros atributos. En cuanto a la predicción de pagos, los resultados de importancia relativa de las variables son los siguientes:

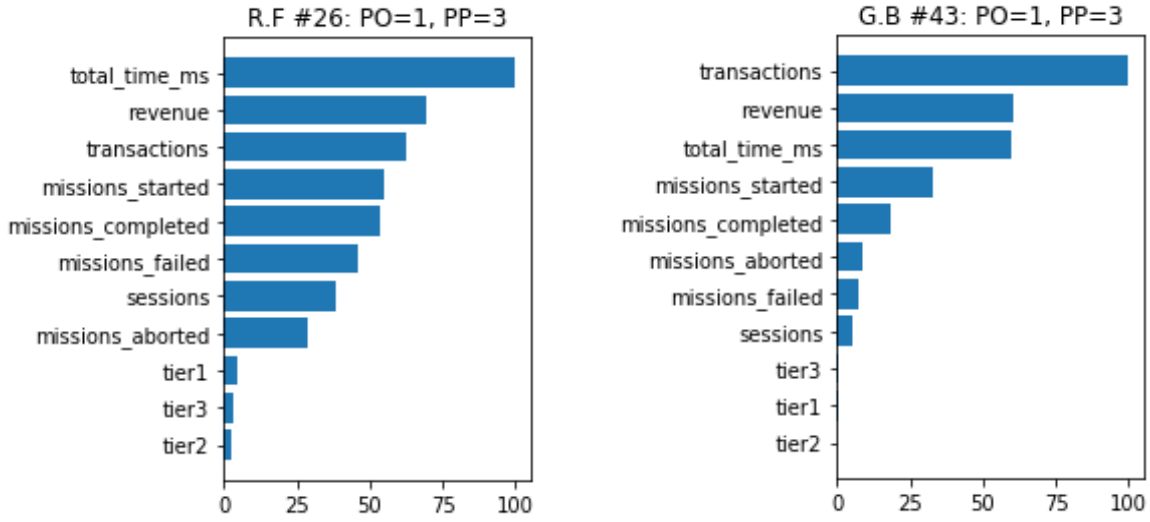

Figura V. Importancia relativa de las variables: pagos a corto plazo. Izquierda: *Random Forest*, derecha: *Gradient Boosting*. Fuente: elaboración propia.

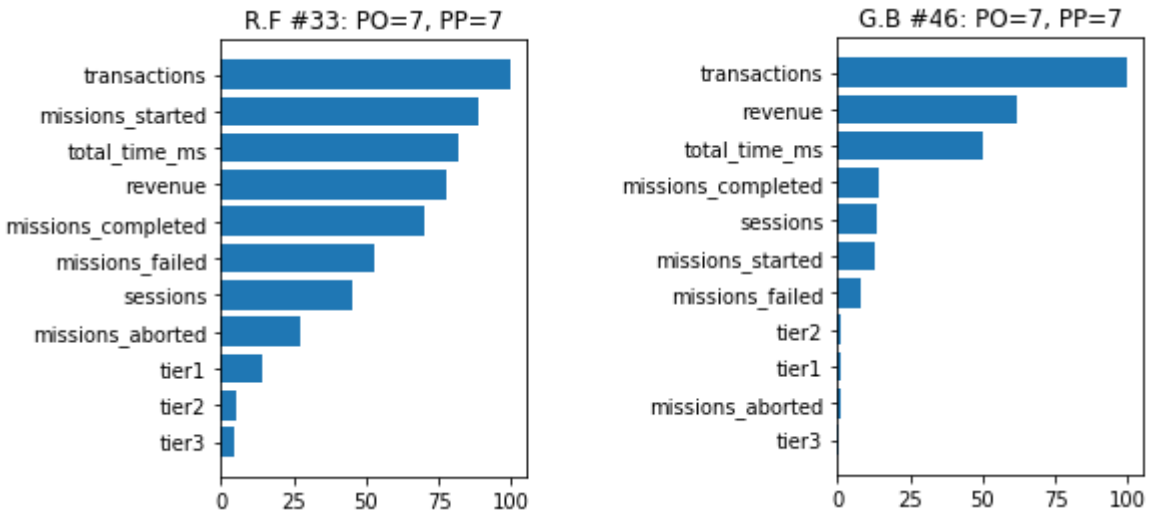

Figura VI. Importancia relativa de las variables: pagos a medio plazo. Izquierda: *Random Forest*, derecha: *Gradient Boosting*. Fuente: elaboración propia.

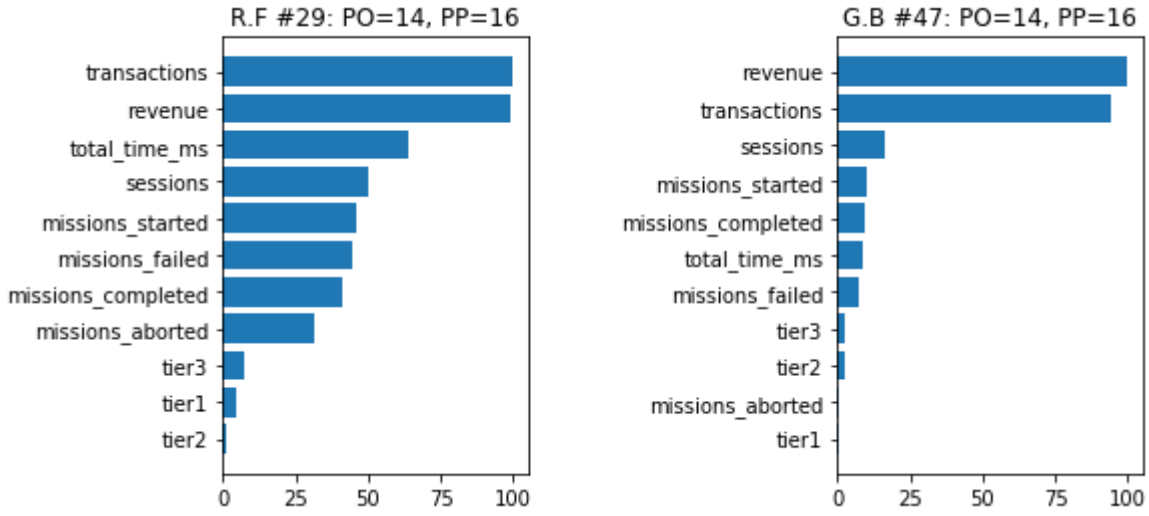

Figura VII. Importancia relativa de las variables: pagos a largo plazo. Izquierda: *Random Forest*, derecha: *Gradient Boosting*. Fuente: elaboración propia.

Tanto para los abandonos como para los pagos, ya sea por *Random Forest* o *Gradient Boosting*, el país de procedencia fue una de las variables de menor importancia relativa. Las variables referentes a pagos e ingresos (*transactions* y *revenue*) resultaron relevantes para la predicción de pagos, pero no para la predicción de abandonos. El resto de variables influyeron en mayor o menor medida en las clasificaciones.

7 Conclusiones

Con el desarrollo del trabajo se concluyó lo siguiente:

1. El análisis de los antecedentes sobre predicción de abandonos y pagadores en videojuegos evidenció que existen métodos para realizar dichas clasificaciones a partir de la información almacenada sobre los usuarios. Sin embargo, sus resultados no son extensibles a "Bubble Word" y sus jugadores, dada la naturaleza variable e irregular de los videojuegos de móviles y su público. Por otro lado, no se encontró en la bibliografía consultada información precisa sobre cómo varía la predicción de abandonos al utilizar distintos períodos de observación.

2. Se predijeron los abandonos y pagos del juego Freemium "Bubble Word" de Genera Games, obteniendo un valor de AUC bueno, para períodos de observación y clasificación a corto, medio y largo plazo.

3. Fue posible predecir tanto abandonos como pagos utilizando el mismo set de variables, a pesar de que estas influyeron de forma distinta en cada predicción.

4. Las variables empleadas (sobre el comportamiento básico del usuario en el juego y el país de procedencia) fueron suficientes para predecir abandonos y pagos de los usuarios, obteniendo un buen AUC.

5. Fue posible predecir, aunque con un grado de aciertos inferior al deseado, usuarios que pagan en una muestra desbalanceada de pagadores (cantidad de pagadores por debajo del 0,5%).

6. Las predicciones fueron logradas utilizando Regresión Logística, Bosques Aleatorios y Potenciación del Gradiente. Cada método aportó buenos resultados en al menos uno de los períodos comprobados.

7. Todo lo anterior fue posible siguiendo el proceso de Descubrimiento de Conocimiento en Bases de Datos: Selección de los datos, Pre-procesamiento, Transformación, Reconocimiento de patrones y Evaluación e interpretación.

8. Con el trabajo se espera que la empresa cuente con un material de guía para futuras predicciones sobre este y otros juegos, además de información puntual sobre los factores que inciden en los abandonos y pagos de sus usuarios.

Referencias bibliográficas

Adjust. (26 de Junio de 2018). *Mobile gaming benchmarks Q1 2018*. Obtenido de Adjust Web site: https://www.adjust.com/blog/mobile-gaming-benchmarks-comparison/

Allison, D. B., Paultre, F., Goran, M. I., Poehlman, E. T., & Heymsfield, S. B. (1995). Statistical considerations regarding the use of ratios to adjust data. *International Journal of Obesity and Related Metabolic Disorders*, 644-652.

Amat Rodrigo, J. (Febrero de 2017). *RPubs - Árboles de predicción: bagging, random forest, boosting y C5.0*. Obtenido de RPubs Web site: https://rpubs.com/Joaquin_AR/255596

Breiman, L. (Octubre de 2018). *RANDOM FORESTS*. Obtenido de Department of Statistics, University of California, Berkeley Web site: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

Brownlee, J. (9 de September de 2016). *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. Obtenido de Machine Learning Mastery Web site: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

Brus Media. (Octubre de 2018). *Country Tier Targeting | Brus Media*. Obtenido de Brus Media Web site: http://www.brusmedia.com/country-tier-targeting/

Buis, M. L. (2012). Stata tip 106: With or without reference. *The Stata Journal*, 162-164.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 321-357.

Coussemen, K., & Van den Poel, D. (2008). Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques. *Expert Systems with Applications* , 313-327.

DeltaDNA. (Octubre de 2018). *Connecting to your data with R - deltaDNA documentation*. Obtenido de DeltaDNA Web site: http://docs.deltadna.com/reference/analyze/direct-sql-access/connecting-to-your-data-with-r/

DeltaDNA. (Octubre de 2018). *DeltaDNA Platform Technology*. Obtenido de DeltaDNA Web site: https://deltadna.com/technology/

DeltaDNA. (Octubre de 2018). *Direct SQL Access - deltaDNA documentation*. Obtenido de Delta DNA Web site: http://docs.deltadna.com/reference/analyze/direct-sql-access/

Drachen, A., El-Nasr, M. S., & Canossa, A. (2013). *Game Analytics: Maximizing the Value of Player Data*. Londres: Springer-Verlag.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 38-54.

Frawley, W. J., Piatetsky-Shapiro, G., & Matheus, C. J. (1992). Knowledge Discovery in Databases: An Overview. *AI Magazine*, 57-70.

Gassó, N., Thuiller, W., Pino, J., & Vilà, M. (2012). Potential distribution range of invasive plant species in Spain. *NeoBiota 12*, 25-40.

Gelman, A. (2008). Scaling regression inputs by dividing by two standard deviations. *STATISTICS IN MEDICINE*, 2865-2873.

Google. (2018). *Comisiones de transacciones - Ayuda de Play Console*. Obtenido de Ayuda de Google Web site: https://support.google.com/googleplay/android-developer/answer/112622?hl=es

Google Cloud. (29 de Junio de 2018). *Crear una plataforma de análisis de juegos móviles: una arquitectura de referencia*. Obtenido de Google Cloud Web site: https://cloud.google.com/solutions/mobile/mobile-gaming-analysis-telemetry

Guitart, A., Pei Chen, P., Bertens, P., & Periáñez, Á. (2017). Forecasting Player Behavioral Data and Simulating in-Game Events. *arXiv*.

Hadiji, F., Sifa, R., Drachen, A., Thurau, C., Kersting, K., & Bauckhage, C. (2014). Predicting Player Chum In the Wild. *IEEE Conference on Computational Intelligence and Games*.

Hernández Pérez, J. F., Cano Gómez, Á. P., & Parra Meroño, M. C. (2016). Taxonomía del videojuego: un planteamiento por géneros. *La pantalla insomne*.

Huang, J., & Ling, C. X. (2005). Using AUC and Accuracy in Evaluating Learning Algorithms. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 299-310.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. New York: Springer.

Kawale, J., Pal, A., & Srivastava, J. (2009). Churn Prediction in MMORPGs: A Social Influence. *Computational Science and Engineering '09 International Conference.* IEEE.

Kim, S., Choi, D., Lee, E., & Rhee, W. (2017). Churn prediction of mobile and online casual games using play log data. *PLoS One 12*.

Koehrsen, W. (Enero de 2018). *Hyperparameter Tuning the Random Forest in Python*. Obtenido de Towards Data Science Web site: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. *R News*, 18-22.

Liftoff. (20 de Agosto de 2018). *NEW! 2018 Mobile Gaming Apps Report*. Obtenido de Liftoff Web site: https://liftoff.io/blog/new-2018-mobile-gaming-apps-report/

Newzoo. (Abril de 2018). *Global Games Market Revenues 2018 | Per Region & Segment*. Obtenido de Newzoo Web site: https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/

Runge, J., Gao, P., Garcin, F., & Faltings, B. (2014). Churn prediction for high-value players in casual social games. *IEEE Conference on Computational Intelligence and Games*. IEEE.

Salas Velasco, M. (1996). La regresión logística. Una aplicación a la demanda de estudios universitarios. *ESTADÍSTICA ESPAÑOLA*, 193-217.

Schapire, R. E. (2003). The Boosting Approach to Machine Learning. An Overview. En D. Denison, M. Hansen, C. Holmes, B. Mallick, & B. Yu, *Nonlinear Estimation and Classification*. Springer.

Scikit-learn. (Octubre de 2018). *Ensemble methods*. Obtenido de Scikit-learn Web site: https://scikit-learn.org/stable/modules/ensemble.html#gradient-tree-boosting

Scikit-learn. (Octubre de 2018). *Scikit-learn: Machine learning in Python*. Obtenido de Scikit-learn Web site: http://scikit-learn.org/stable/

Seif El-Nasr, M., Drachen, A., & Canossa, A. (2013). *Game Analytics*. Londres: Springer.

Sifa, R., Hadiji, F., Runge, J., Drachen, A., Kersting, K., & Bauckhage, C. (2015). Predicting Purchase Decisions in Mobile Free-to-Play Games. *The Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, (págs. 79-85).

Stata. (Octubre de 2018). *Stata: Software for Statistics and Data Science*. Obtenido de Stata Web site: https://www.stata.com/manuals13/rlogit.pdf

Sudhir Patki, P., & Kelkar, V. V. (2013). Classification using Different Normalization Techniques in Support Vector Machine. *International Journal of Computer Applications*.

Tsai, C.-F., & Chen, M.-Y. (2010). Variable selection by association rules for customer churn prediction of multimedia on demand. *Expert Systems with Applications*, 2006-2015.

Verbeke, W., Dejaeger, K., Martens, D., Hur, J., & Baesens, B. (2012). New insights into churn prediction in the telecommunication sector: A profit driven data mining approach. *European Journal of Operational Research*, 211-229.

Viswanathan, P. (Agosto de 2018). *iOS App Store Vs. Google Play Store for App Developers*. Obtenido de Lifewire Web site: https://www.lifewire.com/ios-app-store-vs-google-play-store-for-app-developers-2373130

Watson, H. J., & Wixom, B. H. (2007). The Current State of Business Intelligence. *IEEE Xplore*, 96-99.

Weber, B. (8 de Abril de 2018). *The Platform Evolution of Game Analytics - Towards Data Science*. Obtenido de Towards Data Science Web site: https://towardsdatascience.com/evolution-of-game-analytics-platforms-4b9efcb4a093

Wei, C.-P., & Chiu, I.-T. (2002). Turning telecommunications call details to churn prediction: a data mining approach. *Expert Systems with Applications*, 103-112.

Weiss, G. M. (2004). Mining with Rarity: A Unifying Framework. *Sigkdd Explorations*, Volume 6, Issue 1 - Page 7-18.

Xie, Y., Li, X., Ngai, E., & Ying, W. (2009). Customer churn prediction using improved balanced random forests. *Expert Systems with Applications*, 5445-5449.

Zhu, W., Zeng, N., & Wang, N. (2010). Sensitivity, Specificity, Accuracy, Associated Confidence Interval and ROC Analysis with Practical SAS Implementations. *Northeast SAS User Group proceedings, Section of Health Care and Life Sciences*, (págs. 14-17). Baltimore.

Anexos

Anexo I. Códigos de R y consulta SQL para la extracción y filtrado de los datos.

Ejemplos de extracción de abandonos con Período de Observación = 7 y Período Abandono = 7, y pagadores con Período de Observación = 14 y Período de Pago = 16.

```
library(odbc)
conn <- dbConnect(odbc::odbc(),
                  .connection_string = "Driver={PostgreSQL
Unicode(x64)};Uid=user@domain.com;Pwd=password_here;
                  Server=data.deltadna.net;Port=5432;Database=database_name_here;SslMode=require;"
                  ,bigint = "numeric")

#OP=7 CP=7
stats_churn_query_7_7<-"SELECT user_id, count(session_id) as 'sessions',
    SUM(revenue) AS 'revenue',
    SUM(missions_started) AS 'missions_started', SUM(missions_completed) AS 'missions_completed',
    SUM(missions_aborted) AS 'missions_aborted', SUM(missions_failed) AS 'missions_failed',
    SUM(total_time_ms) AS 'total_time_ms',
    SUM(transactions) AS 'transactions',user_country,
    CASE WHEN user_id IN (
                        SELECT DISTINCT user_id FROM fact_user_sessions_day_live WHERE
DATEDIFF('day',player_start_date,event_date)>7.0
                        AND DATEDIFF('day',player_start_date,event_date)<=14.0
                        AND user_id IN (SELECT user_id FROM user_metrics_live WHERE
((COALESCE(fieldClientVersionFirst,'')='1.2.9'
                                    AND fieldEventTimestampFirst>='2018-07-25 00:00:00.0'
                                    AND fieldEventTimestampFirst<='2018-09-11 00:00:00.0'
                                    AND COALESCE(fieldPlatformFirst,'') LIKE 'ANDROID%'
                                    AND DATEDIFF('day',fieldEventTimestampFirst,NOW())>1.0
                                    AND COALESCE(fieldRevenueValidatedMax,0)<=1.0)))
                    )
    THEN 0 ELSE 1 END AS churn
    FROM fact_user_sessions_day_live
    WHERE (DATEDIFF('day',player_start_date,event_date)<=7.0
    AND user_id in(
           SELECT user_id FROM user_metrics_live WHERE ((COALESCE(fieldClientVersionFirst,'')='1.2.9'
           AND fieldEventTimestampFirst>='2018-07-25 00:00:00.0'
           AND fieldEventTimestampFirst<='2018-09-11 00:00:00.0'
           AND COALESCE(fieldPlatformFirst,'') LIKE 'ANDROID%'
           AND DATEDIFF('day',fieldEventTimestampFirst,NOW())>1.0
           AND COALESCE(fieldRevenueValidatedMax,0)<=1.0)))
    ) GROUP BY user_id,churn ORDER BY user_id"

stats_churn_7_7 <- dbGetQuery(conn, as.character(stats_churn_query_7_7))
write.csv(stats_churn_7_7, file = "Bubble_stats_churn_7_7.csv")

#OP=14 PP=16
stats_payer_query_14_16<-"SELECT user_id, count(session_id) as 'sessions',
    SUM(revenue) AS 'revenue',SUM(missions_started) AS 'missions_started',
    SUM(missions_completed) AS 'missions_completed',SUM(missions_aborted) AS 'missions_aborted',
    SUM(missions_failed) AS 'missions_failed',SUM(total_time_ms) AS 'total_time_ms',
    SUM(transactions) AS 'transactions',user_country,
    CASE WHEN user_id IN (
                        SELECT DISTINCT user_id FROM fact_user_sessions_day_live WHERE
DATEDIFF('day',player_start_date,event_date)>14.0
                        AND DATEDIFF('day',player_start_date,event_date)<=30.0 and revenue>0
                        AND user_id IN (SELECT user_id FROM user_metrics_live WHERE
((COALESCE(fieldClientVersionFirst,'')='1.2.9'
                                    AND fieldEventTimestampFirst>='2018-07-25 00:00:00.0'
                                    AND fieldEventTimestampFirst<='2018-09-11 00:00:00.0'
                                    AND COALESCE(fieldPlatformFirst,'') LIKE 'ANDROID%'
                                    AND DATEDIFF('day',fieldEventTimestampFirst,NOW())>1.0
```

```
                                    AND COALESCE(fieldRevenueValidatedMax,0)<=1.0)))
                  )
    THEN 1 ELSE 0 END AS payer
    FROM fact_user_sessions_day_live
    WHERE (DATEDIFF('day',player_start_date,event_date)<=14.0
    AND user_id in(
           SELECT user_id FROM user_metrics_live WHERE ((COALESCE(fieldClientVersionFirst,'')='1.2.9'
           AND fieldEventTimestampFirst>='2018-07-25 00:00:00.0'
           AND fieldEventTimestampFirst<='2018-09-11 00:00:00.0'
           AND COALESCE(fieldPlatformFirst,'') LIKE 'ANDROID%'
           AND DATEDIFF('day',fieldEventTimestampFirst,NOW())>1.0
           AND COALESCE(fieldRevenueValidatedMax,0)<=1.0)))
    ) GROUP BY user_id,payer,user_country ORDER BY user_id"

stats_payer_14_16 <- dbGetQuery(conn, as.character(stats_payer_query_14_16))
write.csv(stats_payer_14_16, file = "Bubble_stats_payer_14_16.csv")
```

Anexo II. Códigos de Python para el pre-procesamiento y la transformación de los datos.

Ejemplo de pre-procesamiento y transformación del set de pagadores con Período de Observación = 1 y Período de Pago = 3.

```
#PRE-PROCESAMIENTO Y TRANSFORMACION
import pandas as pd

# Leyendo los datos.
# Para cada nuevo dataset solo es necesario insertar su nombre y colocar en el booleano pagadores el
valor True
# si se trata del dataset de pagadores, False en caso contrario. En el booleano sobremuestrear colocar
True
# si se quiere hacer sobremuestreo, False en caso contrario. Size_test indica la proporción de
observaciones que se
# desean en el set de validación. Size_payers indica la proporción de pagadores que se desea en el set de
# entrenamiento, en caso de que se realice sobre-muestreo.
fileName="Bubble_stats_payer_1_3"
pagadores=True
sobremuestrear=True
size_test=0.3
size_payers=0.25

dataset = pd.read_csv(fileName+".csv")

# Quitando columna user_id
dataset=dataset.drop(['user_id'], axis=1)

# Limpiando usuarios con menos de un segundo de tiempo total en el juego
indices_user=dataset[dataset.total_time_ms < 1000].index
dataset = dataset.drop(indices_user)

# Codificando los países por tiers
dataset['tier1']=0
dataset['tier2']=0
dataset['tier3']=0
dataset['country_code']=dataset['user_country']
dataset['country_code']
=dataset['country_code'].map({'FR':1,'US':1,'AU':1,'CA':1,'DK':1,'DE':1,'NL':1,'NO':1,'SE':1,'UK':1,

'AT':2,'BE':2,'CY':2,'CZ':2,'SV':2,'EE':2,'FI':2,'HK':2,'HU':2,'IS':2,

'IE':2,'IL':2,'IT':2,'JP':2,'KW':2,'NZ':2,'NG':2,'PL':2,'PR':2,'SG':2,
                                                'ZA':2,'ES':2,'CH':2,'AE':2})
dataset['tier1']=dataset['country_code'].map({1:1, 2:0})
dataset['tier1'].fillna(0, inplace=True)
dataset['tier2']=dataset['country_code'].map({2:1, 1:0})
```

```
dataset['tier2'].fillna(0, inplace=True)
dataset['tier3']=dataset['country_code'].map({1:0, 2:0})
dataset['tier3'].fillna(1, inplace=True)
dataset=dataset.drop(['country_code'], axis=1)

#Dividiendo la muestra en datos de prueba y datos de entrenamiento
from sklearn.model_selection import train_test_split

if pagadores:
    X_train, X_test, y_train, y_test = train_test_split(dataset.loc[:,'sessions':'tier3'],
dataset['payer'], test_size=size_test, random_state=42)
    X_y_train=pd.DataFrame(X_train)
    X_y_train['payer']=pd.DataFrame(y_train)
    X_y_train['train']=1
    X_y_test=pd.DataFrame(X_test)
    X_y_test['payer']=pd.DataFrame(y_test)
    X_y_test['train']=0
else:
    X_train, X_test, y_train, y_test = train_test_split(dataset.loc[:,'sessions':'tier3'],
dataset['churn'], test_size=size_test, random_state=42)
    X_y_train=pd.DataFrame(X_train)
    X_y_train['churn']=pd.DataFrame(y_train)
    X_y_train['train']=1
    X_y_test=pd.DataFrame(X_test)
    X_y_test['churn']=pd.DataFrame(y_test)
    X_y_test['train']=0

# Sobre-muestreo de datos de entrenamiento en el dataset de pagadores
from sklearn.utils import resample
if pagadores and sobremuestrear:
    payers=X_y_train[X_y_train.payer==1]
    not_payers=X_y_train[X_y_train.payer==0]
    df_payers_upsampled = resample(payers, replace=True,n_samples=int((size_payers*len(X_y_train)-
len(payers))/(1-size_payers)))
    X_y_train = pd.concat([not_payers, df_payers_upsampled])

dataset = pd.concat([X_y_train, X_y_test])
if sobremuestrear:
    dataset.to_csv(fileName+"_train_oversampled.csv")
else:
    dataset.to_csv(fileName+"_train.csv")

# Normalizando los datos
from sklearn import preprocessing
import numpy as np

medias_variables_training={}
std_variables_training={}

variables_scale=['sessions','revenue','missions_started','missions_completed','missions_aborted',
                 'missions_failed','total_time_ms','transactions','tier1','tier2','tier3']

myData_training=dataset.loc[dataset['train']==1]

for i in range(0, len(variables_scale)):
    column_data_train=np.array(myData_training[variables_scale[i]])
    medias_variables_training[variables_scale[i]]=np.mean(column_data_train)
    std_variables_training[variables_scale[i]]=np.std(column_data_train)

dataset_scaled=pd.DataFrame(dataset)
for i in range(0, len(variables_scale)):
    column_data=np.array(dataset[variables_scale[i]])
    dataset_scaled[variables_scale[i]]=(column_data-
medias_variables_training[variables_scale[i]])/std_variables_training[variables_scale[i]]

if sobremuestrear:
    dataset_scaled.to_csv(fileName+"_train_scaled_oversampled.csv")
else:
    dataset_scaled.to_csv(fileName+"_train_scaled.csv")
print 'OK'
```

Anexo III. Códigos de Stata para la Regresión Logística.

Ejemplo de Regresión Logística para el set de abandonos, con Período de Observación = 1 y Período de Abandono = 3.

```
import delimited D:\CSV\Bubble_stats_churn_1_3_train.csv
gen tier=1 if tier1==1
replace tier=2 if tier2==1
replace tier=3 if tier3==1
logit churn sessions missions_started missions_completed missions_aborted missions_failed i.tier if
train==1
lroc if train==0
estat classification if train==0
outreg2 using logit_churn_1_3, append excel dec(7)
```

Anexo IV. Códigos de Python para *Random Forest*, con validación cruzada y optimización de hiperparámetros.

Ejemplo de *Random Forest* para el set de pagadores, con sobre-muestreo, Período de Observación = 14 y Período de Pago = 16.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix

# Leyendo los datos. Para cada nuevo dataset solo es necesario insertar su nombre y colocar
# en el booleano payer el valor True si se trata del dataset de pagadores, False en caso contrario.
dataset = pd.read_csv('Bubble_stats_payer_14_16_train_scaled_oversampled.csv')
payer=True

#Separando en entrenamiento y prueba
myData_training=dataset.loc[dataset['train']==1]
myData_test=dataset.loc[dataset['train']==0]
if payer:
    Y_training=np.array(myData_training['payer'])
    Y_test=np.array(myData_test['payer'])
    X_training= myData_training.drop('payer', axis = 1)
    X_test= myData_test.drop('payer', axis = 1)
else:
    Y_training=np.array(myData_training['churn'])
    Y_test=np.array(myData_test['churn'])
    X_training= myData_training.drop('churn', axis = 1)
    X_test= myData_test.drop('churn', axis = 1)

X_training= X_training.drop('train', axis = 1)
X_test= X_test.drop('train', axis = 1)
X_training=np.array(X_training)
X_test=np.array(X_test)

# Definiendo hiperparámetros
n_estimators = [100,120,150]
max_features = ['sqrt', 'log2']
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

# Diccionario de hiperparámetros
random_grid = {'n_estimators': n_estimators,
```

```
                'max_features': max_features,
                'min_samples_split': min_samples_split,
                'min_samples_leaf': min_samples_leaf,
                'bootstrap': bootstrap}

# Validación cruzada con todas las combinaciones de hiperparámetros
# y distintos tipos de scoring para la selección del mejor resultado
rf = RandomForestClassifier(random_state=42)
grid_search1 = GridSearchCV(estimator = rf, param_grid = random_grid,
                            cv = 3, n_jobs = 6, verbose = 2)
grid_search1.fit(X_training, Y_training)

grid_search2 = GridSearchCV(estimator = rf, param_grid = random_grid,
                             cv = 3, n_jobs = 6, verbose = 2, scoring='roc_auc')
grid_search2.fit(X_training, Y_training)

if payer:
    grid_search3 = GridSearchCV(estimator = rf, param_grid = random_grid,
                                cv = 3, n_jobs = 6, verbose = 2, scoring='balanced_accuracy')
    grid_search3.fit(X_training, Y_training)

# Mejores hiperparámetros
mejores_hiperp1=grid_search1.best_params_
print mejores_hiperp1
mejores_hiperp2=grid_search2.best_params_
print mejores_hiperp2
if payer:
    mejores_hiperp3=grid_search3.best_params_
    print mejores_hiperp3

# Entrenando Random Forest, valores por defecto
rf_default = RandomForestClassifier(random_state=42)
rf_default.fit(X_training, Y_training)
roc_auc=roc_auc_score(Y_test, rf_default.predict_proba(X_test)[:,1])
print 'Observaciones: '+str(len(Y_training))
print 'Área bajo la curva ROC, valores por defecto: '+str(round(roc_auc,4))
predictions_d = rf_default.predict(X_test)
tnd, fpd, fnd, tpd=confusion_matrix(Y_test,predictions_d).ravel()
tpd=float(tpd)
tnd=float(tnd)
accuracy_true_positive_d=tpd/(tpd+fpd)
accuracy_true_negative_d=tnd/(tnd+fnd)
print 'Verdaderos positivos (parámetros por defecto): '+ str(accuracy_true_positive_d*100)
print 'Verdaderos negativos (parámetros por defecto): '+ str(accuracy_true_negative_d*100)

# Entrenando Random Forest, hiperparámetros afinados
rf_tuneado1 = RandomForestClassifier(n_estimators=mejores_hiperp1['n_estimators'],
                                     max_features=mejores_hiperp1['max_features'],
                                     min_samples_split=mejores_hiperp1['min_samples_split'],
                                     min_samples_leaf=mejores_hiperp1['min_samples_leaf'],
                                     bootstrap=mejores_hiperp1['bootstrap'],
                                     random_state=42)
rf_tuneado1.fit(X_training, Y_training)
roc_auc1=roc_auc_score(Y_test, rf_tuneado1.predict_proba(X_test)[:,1])
print 'Área bajo la curva ROC (set de parámetros 1): '+str(round(roc_auc1,4))
predictions1 = rf_tuneado1.predict(X_test)
tn1, fp1, fn1, tp1=confusion_matrix(Y_test,predictions1).ravel()
tp1=float(tp1)
tn1=float(tn1)
accuracy_true_positive1=tp1/(tp1+fp1)
accuracy_true_negative1=tn1/(tn1+fn1)
print 'Verdaderos positivos (set de parámetros 1): '+ str(accuracy_true_positive1*100)
print 'Verdaderos negativos (set de parámetros 1): '+ str(accuracy_true_negative1*100)

rf_tuneado2 = RandomForestClassifier(n_estimators=mejores_hiperp2['n_estimators'],
                                     max_features=mejores_hiperp2['max_features'],
                                     min_samples_split=mejores_hiperp2['min_samples_split'],
                                     min_samples_leaf=mejores_hiperp2['min_samples_leaf'],
                                     bootstrap=mejores_hiperp2['bootstrap'],
                                     random_state=42)
```

```
rf_tuneado2.fit(X_training, Y_training)
roc_auc2=roc_auc_score(Y_test, rf_tuneado2.predict_proba(X_test)[:,1])
print 'Área bajo la curva ROC (set de parámetros 2): '+str(round(roc_auc2,4))
predictions2 = rf_tuneado2.predict(X_test)
tn2, fp2, fn2, tp2=confusion_matrix(Y_test,predictions2).ravel()
tp2=float(tp2)
tn2=float(tn2)
accuracy_true_positive2=tp2/(tp2+fp2)
accuracy_true_negative2=tn2/(tn2+fn2)
print 'Verdaderos positivos (set de parámetros 2): '+ str(accuracy_true_positive2*100)
print 'Verdaderos negativos (set de parámetros 2): '+ str(accuracy_true_negative2*100)

if payer:
    rf_tuneado3 = RandomForestClassifier(n_estimators=mejores_hiperp3['n_estimators'],
                                         max_features=mejores_hiperp3['max_features'],
                                         min_samples_split=mejores_hiperp3['min_samples_split'],
                                         min_samples_leaf=mejores_hiperp3['min_samples_leaf'],
                                         bootstrap=mejores_hiperp3['bootstrap'],
                                         random_state=43)
    rf_tuneado3.fit(X_training, Y_training)
    roc_auc3=roc_auc_score(Y_test, rf_tuneado3.predict_proba(X_test)[:,1])
    print 'Área bajo la curva ROC (set de parámetros 3): '+str(round(roc_auc3,4))
    predictions3 = rf_tuneado3.predict(X_test)
    tn3, fp3, fn3, tp3=confusion_matrix(Y_test,predictions3).ravel()
    tp3=float(tp3)
    tn3=float(tn3)
    accuracy_true_positive3=tp3/(tp3+fp3)
    accuracy_true_negative3=tn3/(tn3+fn3)
    print 'Verdaderos positivos (set de parámetros 3): '+ str(accuracy_true_positive3*100)
    print 'Verdaderos negativos (set de parámetros 3): '+ str(accuracy_true_negative3*100)
```

Anexo V. Códigos de Python para *Gradient Boosting*, con validación cruzada y optimización de hiperparámetros.

Ejemplo de *Gradient Boosting* para el set de pagadores, con sobre-muestreo, Período de Observación = 14 y Período de Pago = 16.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix

# Leyendo los datos. Para cada nuevo dataset solo es necesario insertar su nombre y colocar
# en el booleano payer el valor True si se trata del dataset de pagadores, False en caso contrario.
dataset = pd.read_csv('Bubble_stats_payer_14_16_train_scaled_oversampled.csv')
payer=True

#Separando en entrenamiento y prueba
myData_training=dataset.loc[dataset['train']==1]
myData_test=dataset.loc[dataset['train']==0]
if payer:
    Y_training=np.array(myData_training['payer'])
    Y_test=np.array(myData_test['payer'])
    X_training= myData_training.drop('payer', axis = 1)
    X_test= myData_test.drop('payer', axis = 1)
else:
    Y_training=np.array(myData_training['churn'])
    Y_test=np.array(myData_test['churn'])
    X_training= myData_training.drop('churn', axis = 1)
    X_test= myData_test.drop('churn', axis = 1)

X_training= X_training.drop('train', axis = 1)
X_test= X_test.drop('train', axis = 1)
```

```
X_training=np.array(X_training)
X_test=np.array(X_test)

# Definiendo hiperparámetros
n_estimators = [100,120,150]
max_features = ['sqrt', None]
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
loss = ['deviance', 'exponential']

# Diccionario de hiperparámetros
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'loss': loss}

# Validación cruzada con todas las combinaciones de hiperparámetros
# y distintos tipos de scoring para la selección del mejor resultado
gb = GradientBoostingClassifier(random_state=42)
grid_search1 = GridSearchCV(estimator = gb, param_grid = random_grid,
                            cv = 3, n_jobs = 6, verbose = 2)
grid_search1.fit(X_training, Y_training)

grid_search2 = GridSearchCV(estimator = gb, param_grid = random_grid,
                            cv = 3, n_jobs = 6, verbose = 2, scoring='roc_auc')
grid_search2.fit(X_training, Y_training)

if payer:
    grid_search3 = GridSearchCV(estimator = gb, param_grid = random_grid,
                                cv = 3, n_jobs = 6, verbose = 2, scoring='balanced_accuracy')
    grid_search3.fit(X_training, Y_training)

# Mejores hiperparámetros
mejores_hiperp1=grid_search1.best_params_
print mejores_hiperp1
mejores_hiperp2=grid_search2.best_params_
print mejores_hiperp2
if payer:
    mejores_hiperp3=grid_search3.best_params_
    print mejores_hiperp3

# Entrenando Random Forest, valores por defecto
gb_default = GradientBoostingClassifier(random_state=42)
gb_default.fit(X_training, Y_training)
roc_auc=roc_auc_score(Y_test, gb_default.predict_proba(X_test)[:,1])
print 'Observaciones: '+str(len(Y_training))
print 'Área bajo la curva ROC, valores por defecto: '+str(round(roc_auc,4))
predictions_d = gb_default.predict(X_test)
tnd, fpd, fnd, tpd=confusion_matrix(Y_test,predictions_d).ravel()
tpd=float(tpd)
tnd=float(tnd)
accuracy_true_positive_d=tpd/(tpd+fpd)
accuracy_true_negative_d=tnd/(tnd+fnd)
print 'Verdaderos positivos (parámetros por defecto): '+ str(accuracy_true_positive_d*100)
print 'Verdaderos negativos (parámetros por defecto): '+ str(accuracy_true_negative_d*100)

# Entrenando Random Forest, hiperparámetros afinados
gb_tuneado1 = GradientBoostingClassifier(n_estimators=mejores_hiperp1['n_estimators'],
                                         max_features=mejores_hiperp1['max_features'],
                                         min_samples_split=mejores_hiperp1['min_samples_split'],
                                         min_samples_leaf=mejores_hiperp1['min_samples_leaf'],
                                         loss=mejores_hiperp1['loss'],
                                         random_state=42)
gb_tuneado1.fit(X_training, Y_training)
roc_auc1=roc_auc_score(Y_test, gb_tuneado1.predict_proba(X_test)[:,1])
print 'Área bajo la curva ROC (set de parámetros 1): '+str(round(roc_auc1,4))
predictions1 = gb_tuneado1.predict(X_test)
tn1, fp1, fn1, tp1=confusion_matrix(Y_test,predictions1).ravel()
tp1=float(tp1)
```

```
tn1=float(tn1)
accuracy_true_positive1=tp1/(tp1+fp1)
accuracy_true_negative1=tn1/(tn1+fn1)
print 'Verdaderos positivos (set de parámetros 1): '+ str(accuracy_true_positive1*100)
print 'Verdaderos negativos (set de parámetros 1): '+ str(accuracy_true_negative1*100)

gb_tuneado2 = GradientBoostingClassifier(n_estimators=mejores_hiperp2['n_estimators'],
                                         max_features=mejores_hiperp2['max_features'],
                                         min_samples_split=mejores_hiperp2['min_samples_split'],
                                         min_samples_leaf=mejores_hiperp2['min_samples_leaf'],
                                         loss=mejores_hiperp2['loss'],
                                         random_state=42)
gb_tuneado2.fit(X_training, Y_training)
roc_auc2=roc_auc_score(Y_test, gb_tuneado2.predict_proba(X_test)[:,1])
print 'Área bajo la curva ROC (set de parámetros 2): '+str(round(roc_auc2,4))
predictions2 = gb_tuneado2.predict(X_test)
tn2, fp2, fn2, tp2=confusion_matrix(Y_test,predictions2).ravel()
tp2=float(tp2)
tn2=float(tn2)
accuracy_true_positive2=tp2/(tp2+fp2)
accuracy_true_negative2=tn2/(tn2+fn2)
print 'Verdaderos positivos (set de parámetros 2): '+ str(accuracy_true_positive2*100)
print 'Verdaderos negativos (set de parámetros 2): '+ str(accuracy_true_negative2*100)

if payer:
    gb_tuneado3 = GradientBoostingClassifier(n_estimators=mejores_hiperp3['n_estimators'],
                                         max_features=mejores_hiperp3['max_features'],
                                         min_samples_split=mejores_hiperp3['min_samples_split'],
                                         min_samples_leaf=mejores_hiperp3['min_samples_leaf'],
                                         loss=mejores_hiperp3['loss'],
                                         random_state=42)
    gb_tuneado3.fit(X_training, Y_training)
    roc_auc3=roc_auc_score(Y_test, gb_tuneado3.predict_proba(X_test)[:,1])
    print 'Área bajo la curva ROC (set de parámetros 3): '+str(round(roc_auc3,4))
    predictions3 = gb_tuneado3.predict(X_test)
    tn3, fp3, fn3, tp3=confusion_matrix(Y_test,predictions3).ravel()
    tp3=float(tp3)
    tn3=float(tn3)
    accuracy_true_positive3=tp3/(tp3+fp3)
    accuracy_true_negative3=tn3/(tn3+fn3)
    print 'Verdaderos positivos (set de parámetros 3): '+ str(accuracy_true_positive3*100)

    print 'Verdaderos negativos (set de parámetros 3): '+ str(accuracy_true_negative3*100)
```

Anexo VI. Códigos de Python para representar la importancia relativa de las variables.

Ejemplo para *Gradient Boosting* en el modelo #47.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier

# Leyendo los datos. Para cada nuevo dataset solo es necesario insertar su nombre y colocar
# en el booleano payer el valor True si se trata del dataset de pagadores, False en caso contrario.
dataset = pd.read_csv('Bubble_stats_payer_14_16_train.csv')
payer=True

#Separando en entrenamiento y prueba
myData_training=dataset.loc[dataset['train']==1]
myData_test=dataset.loc[dataset['train']==0]
if payer:
    Y_training=np.array(myData_training['payer'])
```

```
    Y_test=np.array(myData_test['payer'])
    X_training= myData_training.drop('payer', axis = 1)
    X_test= myData_test.drop('payer', axis = 1)
else:
    Y_training=np.array(myData_training['churn'])
    Y_test=np.array(myData_test['churn'])
    X_training= myData_training.drop('churn', axis = 1)
    X_test= myData_test.drop('churn', axis = 1)

X_training= X_training.drop('train', axis = 1)
X_test= X_test.drop('train', axis = 1)
col_names = list(X_training.columns)
X_training=np.array(X_training)
X_test=np.array(X_test)

modelo_tuneado = GradientBoostingClassifier(n_estimators=100,
                                    max_features='sqrt',
                                    min_samples_split=2,
                                    min_samples_leaf=4,
                                    loss='exponential',
                                    random_state=42)


modelo_tuneado.fit(X_training, Y_training)

roc_auc2=roc_auc_score(Y_test, modelo_tuneado.predict_proba(X_test)[:,1])
print 'Área bajo la curva ROC (set de parámetros 2): '+str(round(roc_auc2,4))
predictions2 = modelo_tuneado.predict(X_test)
tn2, fp2, fn2, tp2=confusion_matrix(Y_test,predictions2).ravel()
tp2=float(tp2)
tn2=float(tn2)
accuracy_true_positive2=tp2/(tp2+fp2)
accuracy_true_negative2=tn2/(tn2+fn2)
print 'Verdaderos positivos (set de parámetros 2): '+ str(accuracy_true_positive2*100)
print 'Verdaderos negativos (set de parámetros 2): '+ str(accuracy_true_negative2*100)

# Importancia de variables
import matplotlib.pyplot as plt

feature_importance = modelo_tuneado.feature_importances_
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
col_names=np.array(col_names)
plt.yticks(pos, col_names[sorted_idx])
plt.title('G.B #47: PO=14, PP=16')
plt.show()
```