Data Mining

Trabajo Final

Integrantes

Di Lello, Martin

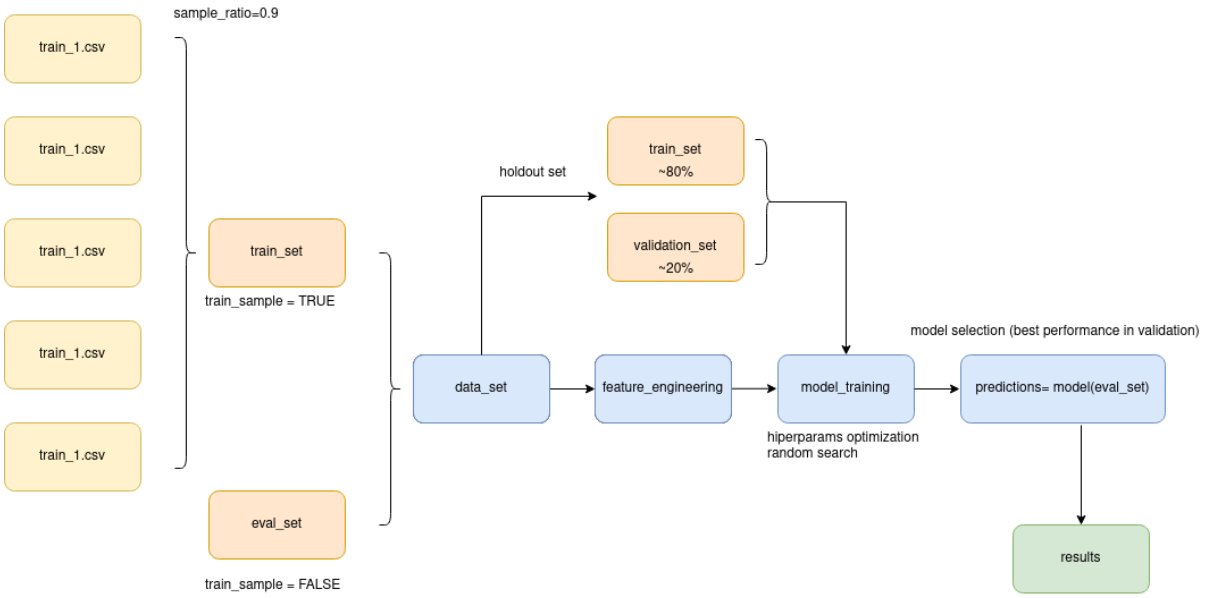
González, Joaquín

Piccardo, Franco

Fecha de entrega: 09 de Mayo de 2021

Introducción

El trabajo consiste en el diseño y desarrollo de un modelo de predicción de churn para el juego ***Castle Crush.*** Se entiende como churn, a aquellos usuarios que luego de jugar al menos una vez durante el tercer día desde la instalación no lo volverán a hacer durante los 14 días posteriores. El pipeline de tareas y etapas que fueron desarrolladas en este trabajo se resumen en la imagen que se adjunta a continuación.



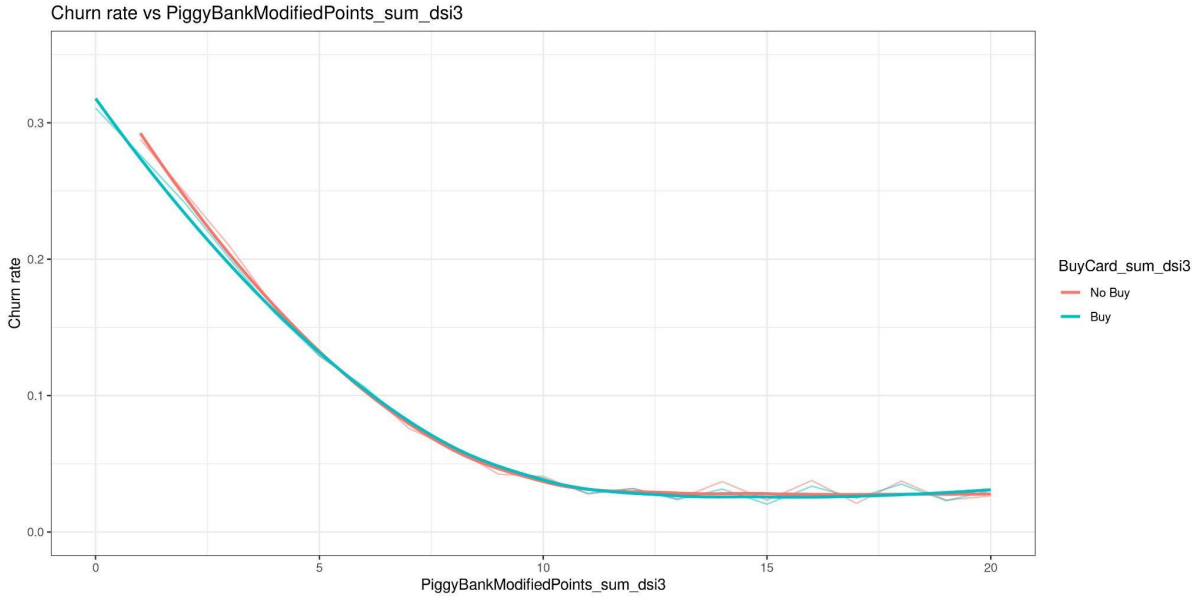
Análisis exploratorio

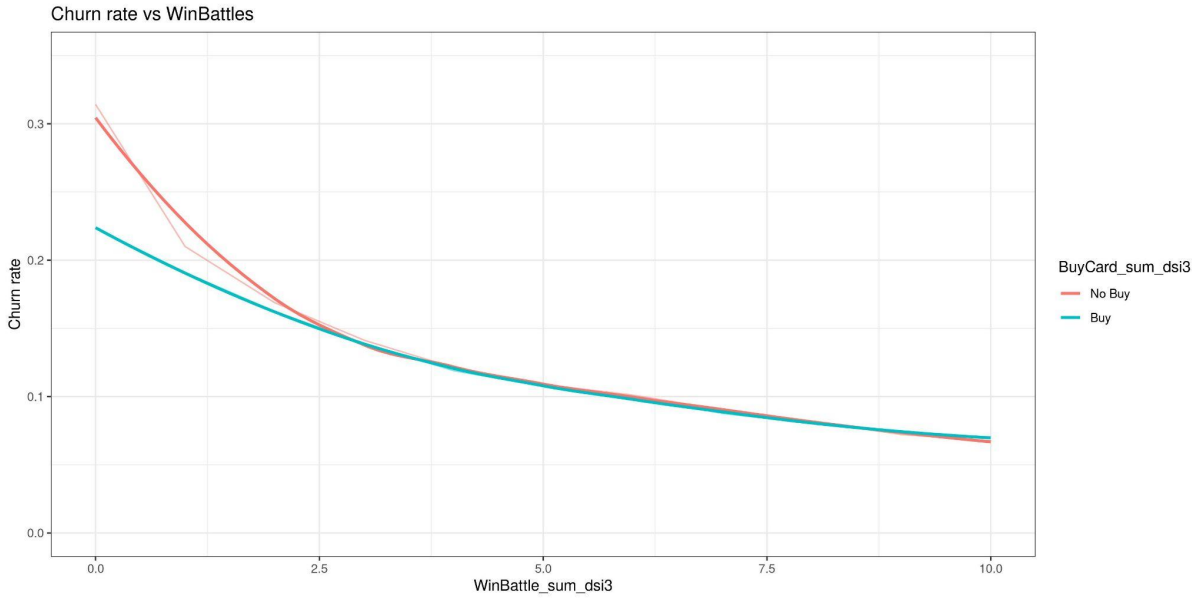
Uno de los primeros desafíos que presenta el trabajo, es encontrar la forma correcta de trabajar con un dataset que tiene millones de registros y más de 100 columnas, que serán incrementadas al hacer one-hot encoding durante la etapa de feature-engineering. Este primer obstáculo se sortea utilizando matrices esparsas que permiten optimizar la forma en la cual esta información es guardada en memoria. Además, se van eliminando los datasets intermedios en runtime y se llama al garbage collector de R para liberar la memoria siempre que sea posible.

El análisis exploratorio lo comenzamos obteniendo métricas básicas como la correlación entre las columnas predictoras del dataset y la columna *Label*. Luego, utilizamos las variables con mayor correlación para realizar gráficos que nos muestren más claramente si había alguna relación lineal/exponencial/otra con churn.

	V1
Label_max_played_dsi	-0.5870106242
PiggyBankModifiedPoints_sum_dsi3	-0.2178906124
WinBattle_sum_dsi3	-0.1754307797
PiggyBankModifiedPoints_sum_dsi2	-0.1711736034
StartBattle_sum_dsi3	-0.1692450204
StartSession_sum_dsi3	-0.1627305762
EnterShop_sum_dsi3	-0.1609722375
PiggyBankModifiedPoints_sum_dsi1	-0.1463387584
LoseBattle_sum_dsi3	-0.1450098926
OpenChest_sum_dsi1	-0.1421974622
StartSession_sum_dsi2	-0.1307449607
WinBattle_sum_dsi2	-0.1303166930
WinBattle_sum_dsi1	-0.1262540780
EnterDeck_sum_dsi1	-0.1229719833
StartBattle_sum_dsi2	-0.1218633031
StartSession_sum_dsi1	-0.1162366322
EnterShop_sum_dsi2	-0.1123868054
EnterDeck_sum_dsi0	-0.1079261741
LoseBattle_sum_dsi2	-0.1041167171
EnterShop_sum_dsi1	-0.1007514944

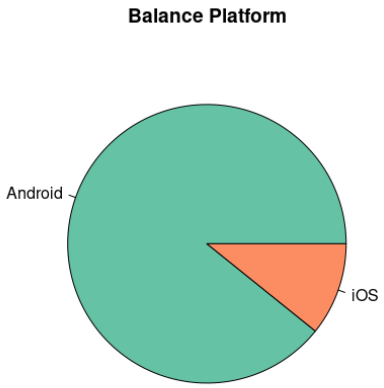
Como resultado de este análisis se obtuvieron las gráficas que se adjuntan a continuación para las variables *PiggyBankModifiedPoints_sum_dsi3* y *WinBattles_sum_dsi3*.

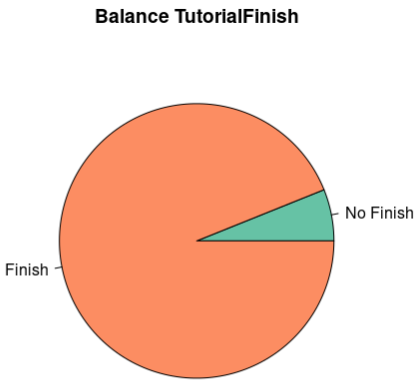




Puede observarse una relación inversa entre el incremento de ambas variables y la evolución del churn rate. Esto nos da indicios de que podrían ser buenas variables predictoras para nuestro caso y que es una buena idea considerarlas para la etapa de feature-engineering.

Además, se analiza el desbalanceo de clases en algunas columnas y la cantidad de missings. Variables como age, que tienen más de un 70% de missings/Nas no son consideradas para el algoritmo de predicción a entrenar.

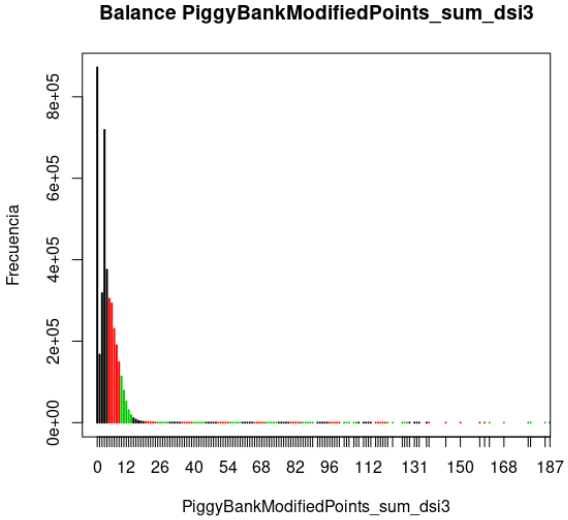


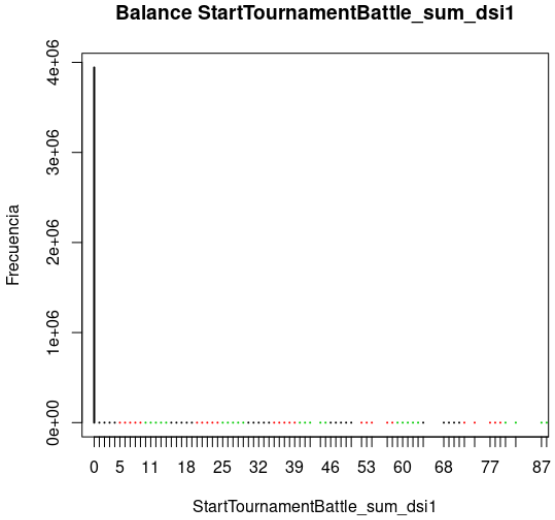


Si hubiésemos considerado KNN como algoritmo final para la competencia, el balanceo y escalamiento de las clases hubiese sido un factor importante a considerar. Como se observa en los gráficos que se adjuntan a continuación,

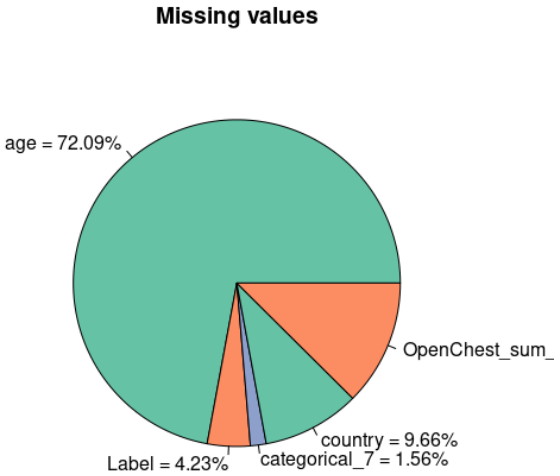
PiggyBankModifiedPoints_sum_dsi3 tiene desbalanceo pero el volumen de valores se

encuentra distribuido entre 0 y 12, mientras que las variables asociadas a *TournamentBattle* tienen más de un 80% de valores 0, por lo que tampoco son contempladas para la etapa de predicción.





Finalmente, analizamos el ratio de NAs para las columnas con más de 100 missings. Como se mencionó anteriormente, *age* fue descartada.



Por último, se observan variables con mucha y con poca varianza que terminaron explicando un gran porcentaje de la varianza de la variable target. No asociamos, a priori, una relación entre varianza y buen candidato de variable predictora en este caso.

Ingeniería de atributos

Fueron utilizados principalmente filtering y wrapper como métodos de selección de atributos. Las variables filtradas inicialmente fueron detalladas en el análisis exploratorio. Luego, se utilizó un proceso iterativo en el cual se fue analizando el feedback que fue devuelto por el algoritmo de boosting para entender cuáles eran las variables con mayor poder predictivo y sobre las cuales hacía sentido generar nuevas columnas. En particular, las nuevas variables diseñadas fueron las siguientes:

Variable	Descripción
max_StartSession_sum	máximo entre variables dsi_[0-3]
min_StartSession_sum	mínimo_entre variables dsi_[0-3]
min_BuyCard_sum	mínimo_entre variables dsi_[0-3]
max_BuyCard_sum	máximo entre variables dsi_[0-3]
min_LoseBattle_sum	mínimo_entre variables dsi_[0-3]
max_LoseBattle_sum	máximo entre variables dsi_[0-3]
max_EnterShop_sum	máximo entre variables dsi_[0-3]
min_EnterShop_sum	mínimo_entre variables dsi_[0-3]
max_WinBattle_sum	máximo entre variables dsi_[0-3]
min_WinBattle_sum	mínimo_entre variables dsi_[0-3]
max_PiggyBankModifiedPoints_sum	máximo entre variables dsi_[0-3]
min_PiggyBankModifiedPoints_su	mínimo_entre variables dsi_[0-3]
max_OpenChest_sum	máximo entre variables dsi_[0-3]
min_OpenChest_sum	mínimo_entre variables dsi_[0-3]
creditor	deudores

Queda pendiente la definición de variables con ratios. Los tiempos de cómputo y la búsqueda, por momentos manual, de los mejores hiperparametros nos requirieron el uso de una gran cantidad de tiempo.

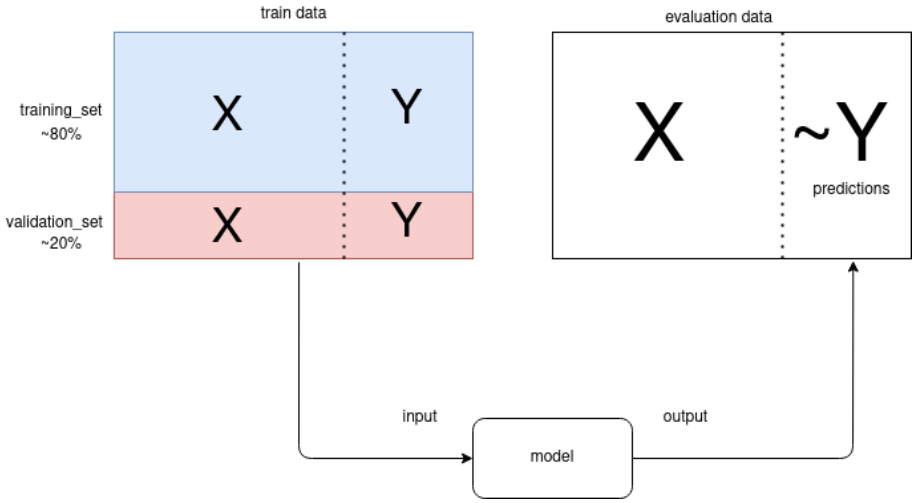
Modelo de validación

Como se muestra en la figura del pipeline al comienzo de este trabajo, se utiliza holdout set como modelo de validación. Se separa del set de entrenamiento aproximadamente un 20% de los datos (considerando un 75% del dataset original por cuestiones de tiempo). Se utiliza este modelo por su simpleza y porque xgboost permite ir seleccionando porciones del

dataset de manera aleatoria a la hora de entrenar cada árbol con los parámetros ***subsample*** (filas) y ***colsample_bytree*** (columnas). De esta manera, la solución conjunta que contempla distintos árboles capturará información de patrones presentes diferentes sectores del dataset. Como métrica de performance del modelo se utilizó AUC.

Mejoras a considerar en este punto, es seleccionar el holdout set de manera aleatoria o implementar un k-fold cross-validation pero esta última opción tendría como desventaja el incremento en tiempo computacional.

La performance obtenida en validación no coincide con la performance obtenida en Kaggle (evaluation_set) ya que el aprendizaje supervisado se realiza con una parte del dataset de entrenamiento y la validación con otra parte del mismo. Pero las predicciones que se realizan sobre el evaluation_set son predicciones sobre información “nueva”, con la que no se entrenó al modelo y sobre la cual no se sabe si los usuarios hicieron o no churn, por lo que es de esperar que haya comportamientos sobre estos datos que nuestro modelo no capture y no pueda representar en sus predicciones.



Algoritmo de aprendizaje

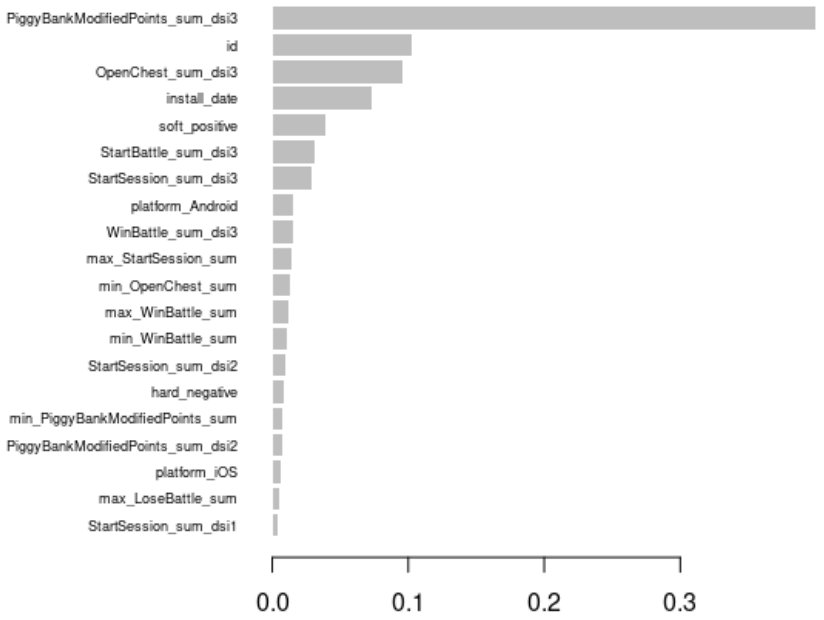
Inicialmente comenzamos con KNN como algoritmo de aprendizaje/predicción, pero debido a la baja performance comenzamos a evaluar otros modelos. El algoritmo utilizado para las predicciones de este trabajo fue **Gradient Boosting** con la implementación de la librería **XGBOOST** . Boosting entrena y ensambla N cantidad de árboles y se centra en la minimización del error de los árboles previos. Debido a este approach, los árboles deben ser construidos de forma secuencial y no pueden paralelizarse, característica que se convierte en una desventaja en términos de performance computacional.
Boosting es un algoritmo que se caracteriza por tener un aprendizaje lento (regulado con el parámetro lambda). La idea es que cada árbol no se especialice, sino que el conjunto sea experto logrando buena predicción. Las características principales son:

- Cada árbol es pequeño y con pocos cortes
- El hiper parámetro lambda ralentiza el aprendizaje
- El modelo ensamble depende del aprendizaje del modelo base

En este trabajo se utilizó ***random search*** como estrategia de optimización de hiper parámetros. La elección se basa en que boosting tiene 7 hiper parámetros, e implementar una estrategia de ***layout search,*** donde la complejidad algorítmica crece de manera exponencial, junto con el volumen de datos del dataset hacen que recorrer todo el espacio de posibles valores sea inviable. ***Random search*** nos permite explorar el espacio de hiper parámetros sin tener que recorrerlo en su totalidad.

Teniendo, para algunos modelos entrenados, overfitting hemos notado mejoras significativas al aumentar la cantidad de árboles (nrounds: 300-600) y la profundidad de los mismos (maxdepth: 3-20). Para compensar el aumento de la profundidad hemos bajado un poco el mínimo de lambda para ralentizar el aprendizaje.

Por último, se adjunta una imagen donde se ven las variables con mayor ganancia en una de las ejecuciones de boosting (aún no había sido removida id). La intuición durante la etapa de exploración de datos fue buena respecto de *PiggyBankModifiedPoints_sum_dsi3* y *WinBattles_sum_dsi3*.



Tiempo

Total de tiempo asignado al trabajo (sin contar tiempo de cómputo): 15hs
Análisis exploratorio: 40%
Ingeniería de atributos: 25%
Selección de modelo y optimización de hiper parámetros: 35%