UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

Trabajo Práctico 2 - Machine Learning

75.06 Organización de datos

Segundo cuatrimestre de 2018

Grupo 33: Los Aristodatos

Alumno:	Goñi, Mauro Ariel
Número de padrón:	87646
Email:	maurogoni@gmail.com

Alumno:	Rolon, Jonathan
Número de padrón:	100137
Email:	rolon.jonathan.ezequiel@gmail.com

Alumno:	Fernandez Pandolfo, Franco
Número de padrón:	100467
Email:	francopandolfo@hotmail.com

Link al repositorio:
https://github.com/JonathanRolon/Organizacion-de-Datos/tree/master/presentacion

1. Introducción

El objetivo de este informe es predecir la probabilidad de que los usuarios de Trocafone realicen la conversión (compra) de un celular en su sitio web a través de algoritmos de machine learning (aprendizaje automatizado). El mismo se dividirá en tres secciones:

- Procesamiento de datos: proceso de selección de datos y cómo realizamos el feature engineering (proceso de adaptación de los datos) para el correcto entrenamiento del algoritmo.
- Algoritmos utilizados: que algoritmos probamos sobre los datos y cuáles fueron los resultados obtenidos con cada uno.
- Conclusión final: cuál fue el algoritmo con el que mejor resultados obtuvimos y porque creemos se logró ese resultado.

Se tuvo en cuenta el análisis realizado en el Trabajo Práctico Número 1 para mejor comprensión y selección de features.

2. Procesamiento de datos

Este informe se encarga de analizar los datos de la plataforma de ecommerce de Brasil de Trocafone, analizando las visitas a su página www.trocafone.com . Trocafone es un side to side Marketplace para la compra y venta de dispositivos electrónicos. Los datos fueron proporcionados en un archivo .csv, en los cuales se encontraba la siguiente información:

● timestamp: Fecha y hora cuando ocurrió el evento. (considerar BRT/ART).
● event: Tipo de evento
● person: Identificador de cliente que realizó el evento.
● url: Url visitada por el usuario.
● sku: Identificador de producto relacionado al evento.
● model: Nombre descriptivo del producto incluyendo marca y modelo.
● condition: Condición de venta del producto
● storage: Cantidad de almacenamiento del producto.
● color: Color del producto
● skus: Identificadores de productos visualizados en el evento.
● search_term: Términos de búsqueda utilizados en el evento.
● staticpage: Identificador de página estática visitada
● campaign_source: Origen de campaña, si el tráfico se originó de una campaña de marketing
● search_engine: Motor de búsqueda desde donde se originó el evento, si aplica.
● channel: Tipo de canal desde donde se originó el evento.
● new_vs_returning: Indicador de si el evento fue generado por un usuario nuevo (New)

o por un usuario que previamente había visitado el sitio (Returning) según el motor de analytics.
● city: Ciudad desde donde se originó el evento
● region: Región desde donde se originó el evento.
● country: País desde donde se originó el evento.
● device_type: Tipo de dispositivo desde donde se genero el evento.
● screen_resolution: Resolución de pantalla que se está utilizando en el dispositivo desde donde se genero el evento.
● operating_system_version: Version de sistema operativo desde donde se origino el evento.
● browser_version: Versión del browser utilizado en el evento

Por otro lado, los siguientes tipos de eventos se encuentran disponibles (en el campo event) sobre los cuales se brinda una breve descripción:
● “viewed product”: El usuario visita una página de producto.
● “brand listing”: El usuario visita un listado específico de una marca viendo un conjunto de productos.
● “visited site”: El usuario ingresa al sitio a una determinada url.
● “ad campaign hit”: El usuario ingresa al sitio mediante una campana de marketing online.
● “generic listing”: El usuario visita la homepage.
● “searched products”: El usuario realiza una búsqueda de productos en la interfaz de búsqueda del site.
● “search engine hit”: El usuario ingresa al sitio mediante un motor de búsqueda web.
● “checkout”: El usuario ingresa al checkout de compra de un producto.
● “staticpage”: El usuario visita una página
● “conversion”: El usuario realiza una conversión, comprando un producto.
● “lead”: El usuario se registra para recibir una notificación de disponibilidad de stock, para un producto que no se encontraba disponible en ese momento.

Para este trabajo en particular se tiene 3 archivos:
- events_up_to_01062018.csv
- labels_training_set.csv
- trocafone_kaggle_test.csv

El archivo events_up_to_01062018.csv contiene en el mismo formato detallado anteriormente información de eventos realizado en la plataforma para un conjunto de usuarios hasta el 31/05/2018.

Por otro lado el archivo labels_training_set.csv indica para un subconjunto de los usuarios incluidos en el set de eventos events_up_to_01062018.csv si los mismos realizaron una conversión (columna label = 1) o no (columna label = 0) desde el 01/06/2018 hasta el 15/06/2018.

Se pide indicar la probabilidad de conversión para usuarios que no se encuentran en el archivo labels_training_set.csv , pero para los cuales se cuenta con información en events_up_to_01062018.csv. El listado de estas personas está provisto en el archivo trocafone_kaggle_test.csv.

Estructura del proyecto: **presentación** (folder)

Notebooks procesados: *(ver en carpeta Ganador >> Preprocesados…)*

features_num_cat_first.ipynb : contiene la cantidad de features que se fueron armando numéricos y categóricos, al inicio del procesamiento.

Generando_csv_num_cat.ipynb : realizamos el join de los features numéricos reemplazando los nulos por ceros, "mergeando" por mes: enero, febrero,….,mayo

Los siguientes tres notebooks contienen los join finales en el dataframe o matriz a utilizar para train o test:

FUNCIONES_CATEGORICOS_FINAL_TRAIN-1-.ipynb: Genera los features categóricos para train, utilizando Feature Hashing.

FEATURES_CATEGORICOS_FINAL_TEST-2-.ipynb: Genera los features categóricos para test, utilizando Feature Hashing.

FEATURES_CATEGORICOS_FINAL_SIN-NULOS-TRAIN-TEST-3-.ipynb: Elimina las columnas nulas de los features categóricos de train y de test, para evitar que las matrices sean tan dispersas innecesariamente.

A propósito: *no* se utilizó *mean encoding* ya que el archivo trocafone de usuarios a predecir no contiene labels, cómo sacar un promedio en base a labels sin labels? en base a que? en este caso decidimos optar por feature hashing que si bien genera muchas colisiones si utilizamos pocas instancias, generando mayores instancias da muy buenos resultados.
Porque no usar One Hot Encoding? o binary encoding? no somos muy amantes de los números simplistas binarios… En realidad tenemos entendido que casi nunca funciona pero es muy utilizado. Hay otras maneras de encoding, muchas...

checkpoints (folder): esta carpeta contiene todos los notebooks con todos los algoritmos probados, si usted dispone de tiempo, puede juzgar nuestra labor aquí.

distintos_features (folder): esta carpeta contiene los features utilizados anteriormente sumados algunos features más que fallaron en el intento de mejorar la última probabilidad alcanzada por XGBOOST, básicamente se conforman de notebooks que tienen información parecida a la que veníamos manejando pero agregando o quitando sutilezas que al final no aportaron demasiado.

test.zip y **train.zip** son archivos de 200 mb generados para testear y entrenar que por razones obvias no están descomprimidos.

A partir de esta información y el análisis realizado en el trabajo práctico número uno sobre los datos de se separaron y prepararon los features tanto numéricos como categóricos para finalmente prepararlos de diferentes maneras y unirlos todos para probarlos con diferentes algoritmos de machine learning. Para entrenar los algoritmos siempre se deben usar datos viejos, mientras que para testearlos se deben usar los datos más nuevos.

2.1. Features Numéricos

Para los datos numéricos se fue separando y agrupando el data frame para luego hacer un merge de todos los datos.

Se agrupó a los datos por usuario, por mes y por un tercer dato que fue seleccionado según el análisis realizado en el TP1 y lo que creíamos podría influir de alguna manera en la predicción.

Cabe aclarar que se tuvieron en cuenta varios modelos de features hasta quedarnos con los que aportaron al algoritmo ganador (antes y luego de probarlo para intentar refinar el porcentaje final), a continuación se describen los modelos: Cantidad de eventos por usuario: este feature permite quedarnos con la dimensión intrínseca de person que es de 19414 en el set de labels y de 19415 en el set de trocafone-kaggle, por este motivo decidimos agregarlo.

- cantidad de checkouts por user por mes: es claro que este feature tiene peso como cantidad, por ese motivo y porque influye en la conversión (checkout es el paso anterior a realizar una conversión) decidimos incluirlo como cantidad.

- cantidad de conversiones: son las conversiones pasadas es decir estamos prediciendo para el mes de junio pero para el train se tomaron estas conversiones en cantidad, si bien la cantidad de personas que convierte en un periodo suele no volver a convertir la misma en el periodo siguiente, pensamos podría aportar algún peso en el caso de que hubieran vuelto a convertir en el periodo de junio.

- cantidad de leads por mes: la cantidad de leads por mes aporta en el sentido de que si un usuario tiene más cantidad entonces debería tener una mayor probabilidad de compra para tal producto.

- cantidad de staticPagesVisitadas: el indicador de página estática indica cuántas veces el user se dirigió a la home page, lo incluimos como posible feature aunque no pareciera aportar demasiado por sí solo.

- cantidad de productos vistos: la cantidad de productos vistos puede influir en el usuario en una compra futura

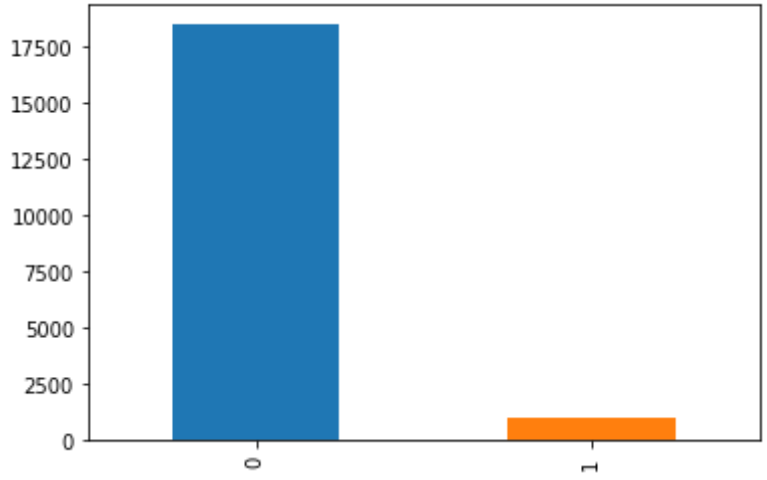
- cantidad de listas de productos vistas: idem anterior, la cantidad de listados vistos influye en el usuario para una compra reciente o futura

Otros features numéricos utilizados:

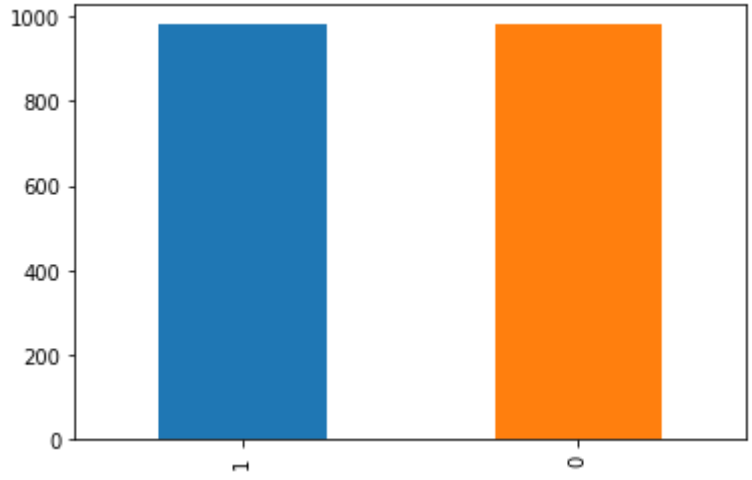
- cantidad de recurrencias del usuario
- cantidad de ingresos al sitio mediante una campaña de marketing
- cantidad de ingresos al sitio mediante motores de búsqueda
- cantidad de veces que se visita la home page

También se buscaron las cantidades promedio de cada feature. Para algunos de los algoritmos(la mayoria) se decidió hacer un undersample para equilibrar la cantidad de unos y ceros obteniendo los siguientes resultados:

Datos sin undersample:



Datos con undersample:



2.2. Features Categóricos

Para los datos categóricos se agrupó al data frame por evento, para luego agruparlos por usuario, mes y alguna dato categórico. Se utilizó Feature Hashing[1] y Mean Encoding[2] para pasar los datos categóricos a numéricos. Para transformar los usuarios se uso tambien one hot encoding[3].

los datos categóricos usados fueron los siguientes:

- Color: para el color se usaron los eventos productos vistos, conversiones y checkouts.
- Condición: para la condición se usaron los eventos checkouts, conversiones y productos vistos.
- Modelo: para el modelo se usaron los eventos productos vistos, checkouts y conversiones.
- Skus: para los skus se usaron los eventos productos buscados, visitas a la home page y visitas a listados de productos.
- Términos buscados: para los términos buscados se usaron los eventos productos buscados.
- Resolución de pantalla: para la resolución de pantalla se usaron los eventos visitas al sitio.
- Almacenamiento: para almacenamiento se usaron los eventos conversiones.
- Origen de campaña: para el origen de campaña se usaron los eventos ingreso mediante campaña de marketing.
- País: para el país se usaron los eventos visitas al sitio.
- Ciudad: para la ciudad se usaron los eventos visitas al sitio.
- Región: para la región se usaron los eventos visitas al sitio.

[1] basado en el uso de una función de hashing para determinar el número de columna de cada dato. Usando feature-hashing podemos definir la cantidad de columnas que queremos usar: m y luego simplemente aplicar a cada valor de las variables categóricas una función de hashing que devuelva un número entre 0 y m - 1 incrementando dicha posición.

[2] sistema que toma en cuenta los labels y las variables para transformar las labels a valores comprensibles para machine learning. El encoding se calcula con el número de variables true sobre el total de variables.

[3] consiste en dividir el atributo en tantas columnas como valores posibles puede tener y usar cada columna como un dato binario indicando si el atributo toma o no dicho valor.

- Tipo de dispositivo: para el tipo de dispositivo se usaron los eventos visitas al sitio, checkouts y conversiones.

Para los features categóricos también se buscó filtrar sobre los n datos más importantes de cada feature. Otro método fue separar cada evento con sus features para después unir todos en un data frame.

Aclaración: no usamos ventanas deslizantes en el armado de la matriz porque se nos complicaba con los features categóricos, cuando "dropeabamos" creabamos un dataframe con una cantidad de personas inferior lo cual no era rentable para calcular probabilidades.
El armado fue el siguiente: tener features desde enero a mayo como columnas ordenados para que al momento de realizar el split, se pueda seleccionar el mes de mayo como test o un porcentaje de mayo, o también un porcentaje de abril ademas de mayo.

Métodos utilizados para el manejo de features (cantidades, dimensiones, importancia)

2.3. SVD o Descomposición en Valores Singulares

La SVD (Singular Value Decomposition) es una de las herramientas más importantes para la reducción de dimensiones de un set de datos y surge de realizar una descomposición de una matriz. La SVD está íntimamente relacionada con el hecho de diagonalizar una matriz simétrica.

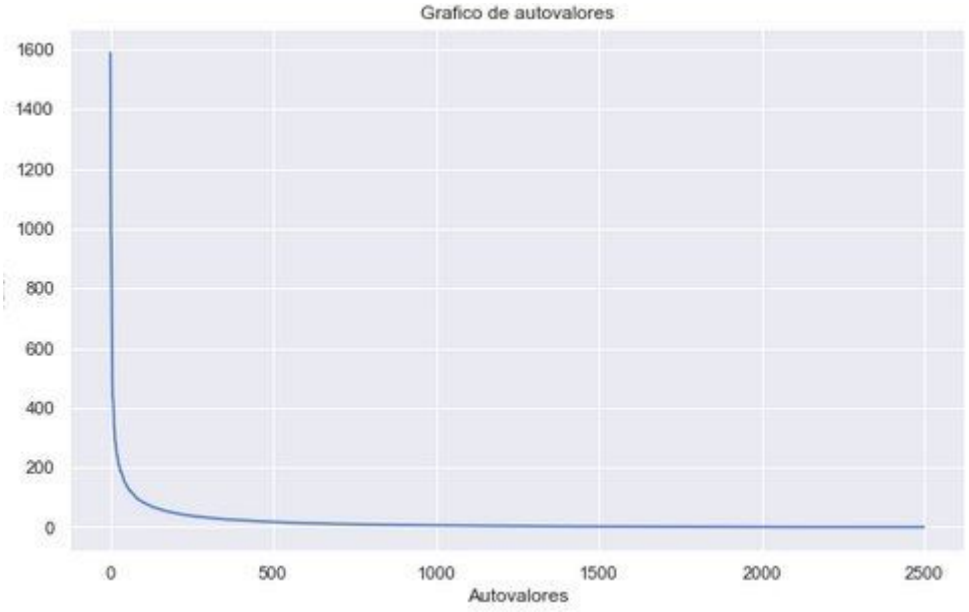
Sea A una matriz de nxm, se puede demostrar que existen tres matrices U, V y E tales que:
A = U E V t

Donde:

U es una matriz unitaria en nxn.
V es otra matriz unitaria en mxm.
y E es una matriz diagonal en nxm.

A continuación, se muestra un método para obtener dichas matrices:
1. Calcular los autovalores y autovectores de AtA.
2. La matriz E se compone de la raíz cuadrada de los autovalores de AtA de forma diagonal y ordenados de forma descendiente.
3. La matriz U se compone con los autovectores normalizados como columnas.
4. La matriz V se puede obtener resolviendo el sistema AV = U sabiendo que V T V es la matriz identidad.

Aplicando dicha herramienta a nuestro set de datos, podemos observar que la dimensionalidad de nuestros datos se encuentra por debajo de 500.



Si nos apoyamos en la energía de la matriz, observamos que la misma acumula un 99% con sus 392 autovalores más representativos, por lo que concluimos que la verdadera dimensión de nuestros datos se encuentra alrededor de dicho valor.

```
Number of Singular Values 377: 24.690702 98.92%
Number of Singular Values 378: 24.633732 98.92%
Number of Singular Values 379: 24.521103 98.93%
Number of Singular Values 380: 24.472889 98.93%
Number of Singular Values 381: 24.326979 98.94%
Number of Singular Values 382: 24.228193 98.94%
Number of Singular Values 383: 24.123617 98.95%
Number of Singular Values 384: 24.082722 98.95%
Number of Singular Values 385: 24.052294 98.96%
Number of Singular Values 386: 23.979631 98.96%
Number of Singular Values 387: 23.923409 98.97%
Number of Singular Values 388: 23.820484 98.98%
Number of Singular Values 389: 23.76509 98.98%
Number of Singular Values 390: 23.732004 98.99%
Number of Singular Values 391: 23.548868 98.99%
Number of Singular Values 392: 23.511427 99.0%
```

2.4. Cross Validation

El proceso de K-fold Cross Validation comienza particionando el set de entrenamiento en k bloques. Luego vamos a realizar varias iteraciones en las cuales entrenamos nuestro algoritmo con k - 1 bloques y lo validamos con el restante. Este proceso se repite k veces para que todos los datos hayan participado alguna vez del set de validación. El resultado es el promedio de las k iteraciones del algoritmo. Esto hay que hacerlo por cada valor posible para nuestros hiperparametros, por lo que, dependiendo de los datos, puede resultar un proceso costoso, pero de dicha forma nos aseguramos que el muestreo tomado, sea aceptable y no descartemos opciones producto de una situación azarosa, ya que es posible que la combinación elegida no funcione como esperábamos, y sea descartada una opción favorable, o en contrapartida, que mantengamos una opción favorable cuando esta no lo es en realidad. Hemos utilizado esta herramienta para realizar el tuning de los hiperparametros de XGBOOST a fin de mejorar las predicciones.

3. Confusion Matrix

Una matriz de confusión es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases.

4. Algoritmos utilizados

A fin de poder contar con el algoritmo que nos permita predecir de mejor manera, hemos decidido probar los siguientes algoritmos:

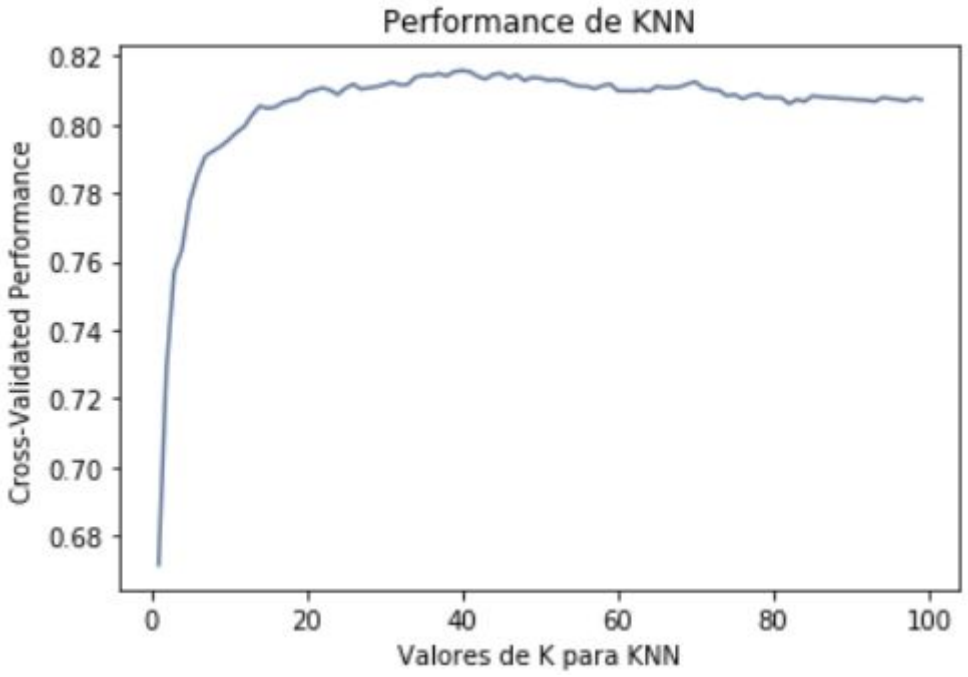
4.1. Logistic Regression

Hemos usado este tipo de análisis de regresión para predecir el resultado de una variable binaria (Convirtió o No-Convirtio) en función de las variables independientes o predictoras. Es útil para modelar la probabilidad de un evento ocurriendo en función de otros factores. También optamos por este algoritmo dada la simplicidad y poco costo computacional que propinaba para su ejecución.

Se obtuvo un score de 0.71030.

4.2. KNN

Este algoritmo clasifica cada dato nuevo en el grupo que corresponda, según tenga k vecinos más cerca de un grupo o de otro. Es decir, calcula la distancia del elemento nuevo a cada uno de los existentes, y ordena dichas distancias de menor a mayor para ir seleccionando el grupo al que pertenece. Hemos optado por probar este algoritmo dado que no cuenta con una fase de entrenamiento, pero no ha tenido la mejor performance, así mismo para la simplicidad del algoritmo ha tenido muy buena performance. Se ha probado para varios valores de k, obteniendo su mejor resultado para K=40, tal como se aprecia a continuación.



Se obtuvo un score de 0.78168.

4.3. GaussianNB

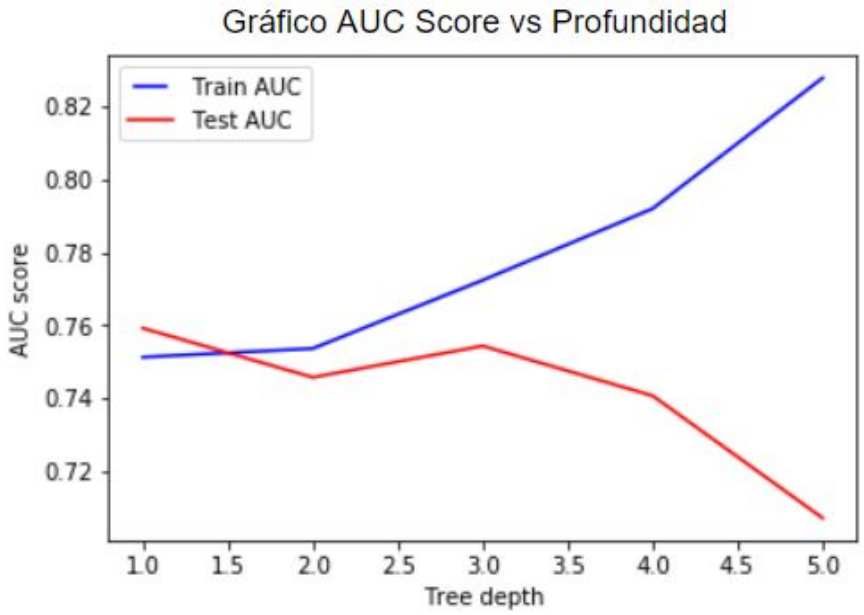
Algoritmo basado en el teorema de Bayes. El teorema básicamente examina la probabilidad de un evento basados en conocimiento anterior de cualquier evento relacionado con el evento a examinar. Se verificó y ya en las pruebas realizada a priori, se descartó el mismo, dado que su performance no era la suficiente.

Se obtuvo un score de 0.46339.

4.4. Árbol de decisión

Un árbol de decisión es un árbol binario en donde en cada nodo dividimos el set de datos en dos de acuerdo a un cierto criterio. El objetivo es llegar a nodos hoja en los cuales podamos clasificar correctamente nuestros datos.

Se obtuvo el siguiente gráfico según la profundidad del árbol:



Dado la relación en el gráfico precedente, se tomó como profundidad para el algoritmo 3, ya que de esta forma se obtuvo los mejores valores en la fase de test.

Se obtuvo un score de 0.79049.

4.5. SVM

SVM es otro algoritmo de clasificación lineal, es decir que la salida del mismo será un vector n-dimensional. Una recta en dos dimensiones o un hiperplano en n-dimensiones. El mejor hiperplano es aquel que maximiza el margen entre las clases. Es decir el que logra la máxima separación entre las clases. De todos los separadores lineales posibles hay solo uno que cumple con esta condición. Se seleccionó este algoritmo dado que queríamos separar en 2 clases, aquellas persona que convierten y aquellas que no. Pero en la fase de prueba no obtuvo los resultados esperados por lo que no se continuó con el mismo.

4.6. Linear Discriminant Analysis

Este algoritmo encuentra una combinación lineal de los features para separarlos en clases.

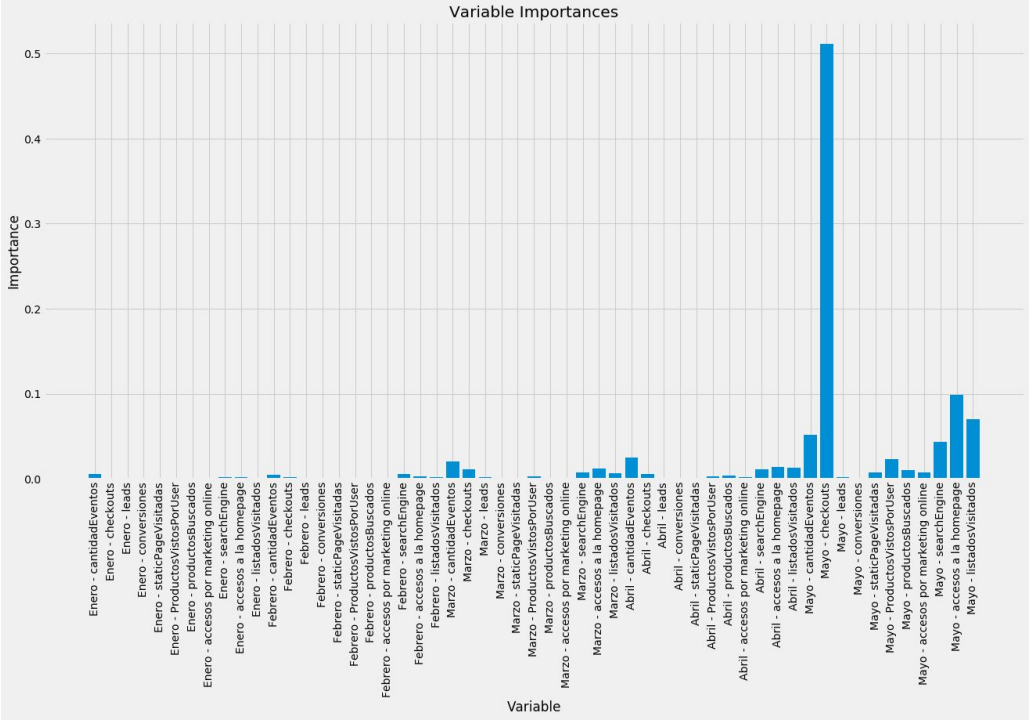
El objetivo de este algoritmo es reducir las dimensiones de los datos. Luego de aplicar el algoritmo, los features minimizan la distancia entre muestras de la misma clase, mientras que maximizan la distancia entre muestras de clases diferentes. Dado que la performance fue bastante inferior a otros en la fase de desarrollo se procedió a descartar este algoritmo.

4.7. Random Forest

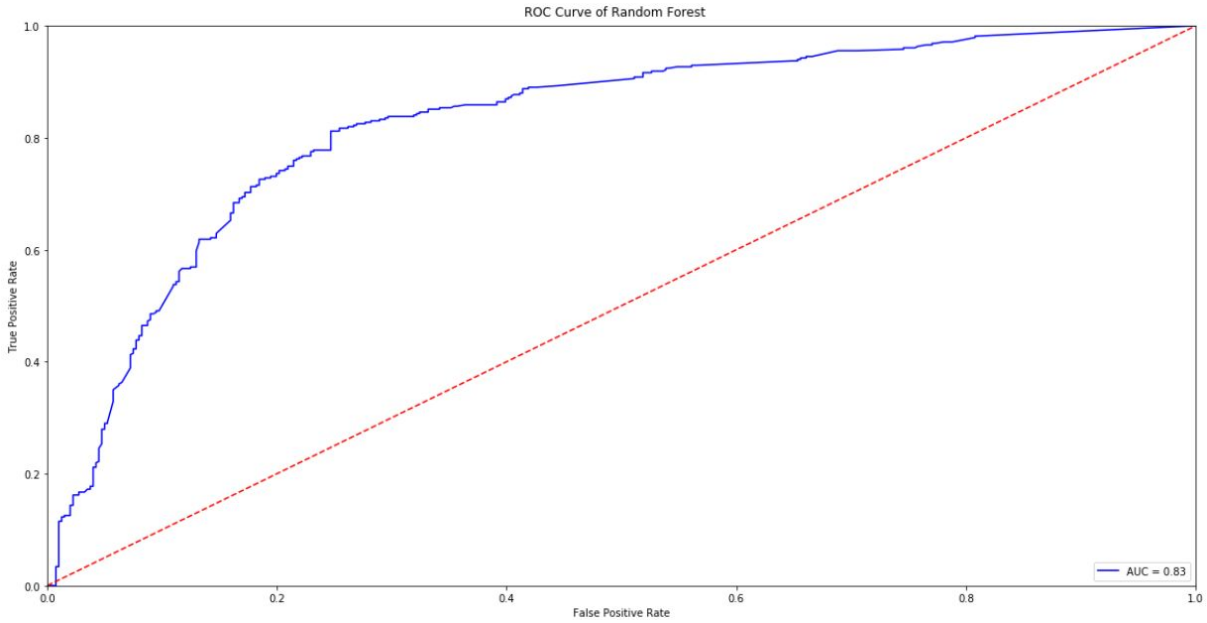
En un Random Forest obtenemos un conjunto de n árboles de decisión en donde cada árbol intenta clasificar a nuestros puntos. Tomando dos puntos podemos calcular su distancia como el número de árboles en los que RF predice una clase diferente para cada punto. Normalizando sobre la cantidad de árboles n tenemos una distancia entre 0 y 1.

Se aplicó la función log para evitar valores extremos en los features, luego se normalizaron los datos y se verificó que no haya NAN.

El siguiente gráfico muestra la importancia que el algoritmo de Random Forest le dio a los datos:



Se realizó un gráfico de curva ROC sobre las predicciones en la etapa de entrenamiento dando como resultado la siguiente gráfica:



El score obtenido fue de 0.82761.

4.8. XGBoost

XGBoost es el algoritmo estado del arte en clasificación basado en potenciación del gradiente[4] con árboles de decisión.

Para la utilización de este algoritmo de ML se utilizó el archivo de features generados para train y test, se realizó un undersample de la muestra dejando una relación de 9 a 1 entre “No conversiones” y “Conversiones”, lo que creímos conveniente dado que la proporción de las No conversiones generalmente es mayor. Se decidió tomar dicha relación, a fin de agregar mayor información al set de datos. Al correr el algoritmo con parámetros seteados de forma arbitraria vimos una mejora en la predicción que él mismo realizaba por lo que nos abocamos a realizar el tuneo de los mismos; se utilizó gridsearch[5] con scoring='roc_auc', donde se realizaron distintas pruebas sobre los diferentes parámetros:

1. 'max_depth': Profundidad máxima de un árbol. El aumento de este valor hace que el modelo sea más complejo y que se sobreajuste con más probabilidad. 0 indica que no hay límite. Valor predeterminado: 6.
2. 'min_child_weight': Suma mínima de la ponderación de instancias (hessiana) necesaria en un elemento secundario. Si el paso de partición del árbol genera un nodo de hoja con la suma de la ponderación de instancia inferior a min_child_weight, el proceso de creación deja de realizar la partición. En los modelos de regresión lineal, esto simplemente se corresponde con un número mínimo de

[4] es una técnica de aprendizaje automático utilizado para el análisis de la regresión y para problemas de clasificación estadística, el cual produce un modelo predictivo en forma de un conjunto de modelos de predicción débiles.

[5] método utilizado para la optimización de hiperparametros. Crea una cuadrícula de hiperparametros y para cada combinación entrena un modelo y le da un puntaje. Se prueban todas las combinaciones de hiperparametros posibles.

instancias necesarias en cada nodo. Conforme mayor sea el algoritmo, más conservador será. Valores válidos: número flotante. Rango: [0,∞). Valor predeterminado: 1.

3. 'gamma': La reducción de pérdida mínima necesaria para realizar una partición adicional en un nodo hoja del árbol. Cuanto mayor es el valor de gamma, más conservador será el algoritmo. Valores válidos: número flotante. Rango: [0,∞). Valor predeterminado: 0.
4. 'subsample': La proporción de la submuestra de la instancia de capacitación. Si se establece en 0,5, significa que *XGBoost* recopila de forma aleatoria la mitad de las instancias de datos para que los árboles crezcan. Esto evita el sobreajuste. Valores válidos: número flotante. Rango: [0,1]. Valor predeterminado: 1.
5. 'colsample_bytree': Proporción de la submuestra de columnas cuando se construye cada árbol. Valores válidos: número flotante. Rango: [0,1]. Valor predeterminado: 1
6. 'reg_alpha': Término de regularización L1 en ponderaciones. Al aumentar este valor, el modelo será más conservador. Valor predeterminado: 1.

En primera instancia se determinó la cantidad ‘n_estimators’ a utilizar con la realización de 2 pruebas, donde se tomó el promedio de realizar cross-validation con 5 contenedores:

([mean: 0.84487, std: 0.00803, params: {'n_estimators': 50},
  mean: 0.84580, std: 0.00821, params: {'n_estimators': 100},
  mean: 0.84460, std: 0.00740, params: {'n_estimators': 150},
  mean: 0.84312, std: 0.00704, params: {'n_estimators': 200},
  mean: 0.83979, std: 0.00635, params: {'n_estimators': 250},
  mean: 0.83730, std: 0.00626, params: {'n_estimators': 300},
  mean: 0.83583, std: 0.00677, params: {'n_estimators': 350}],
 {'n_estimators': 100}, 0.8458020963487437)

Luego se trabajó sobre los parámetros 'max_depth' y 'min_child_weight' determinando los siguientes valores: 'max_depth'=3 y 'min_child_weight'=7.
{'max_depth': 3, 'min_child_weight': 7}, 0.8477411032440187)

Se continuó con la optimización de 'gamma' quedando esta en 'gamma'=0.
{'gamma': 0.0}, 0.8477411032440187)

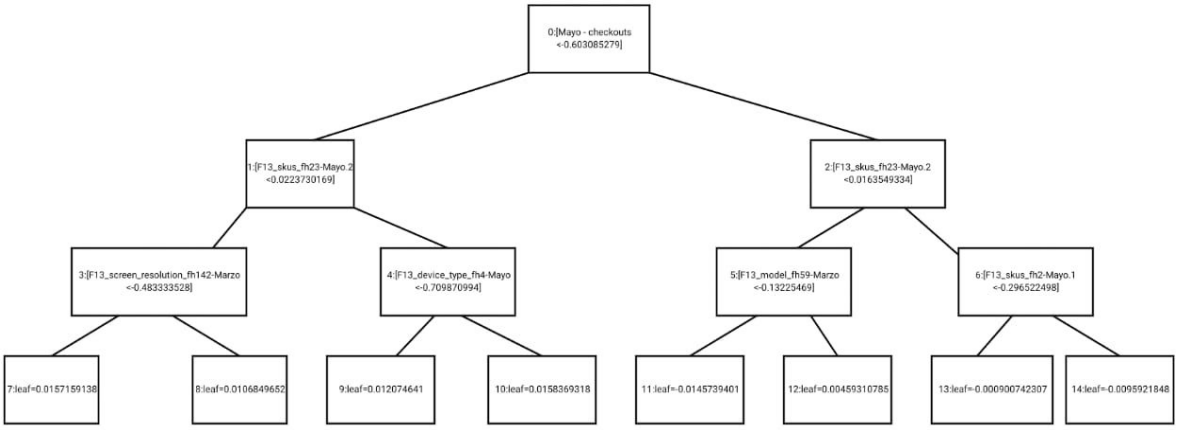
Posteriormente se siguió con 'subsample' y 'colsample_bytree', quedando ambas con los siguientes valores: 'subsample'=0.8 y 'colsample_bytree'=0.3.
{'colsample_bytree': 0.3, 'subsample': 0.8}, 0.8481275163126476)

Finalmente se obtuvieron los valores para 'n_estimators'=800 y 'learning_rate'=0.01. y de 'reg_alpha' quedando la misma en 'reg_alpha'=0.011.
{'reg_alpha': 0.011}, 0.8486842866398259)

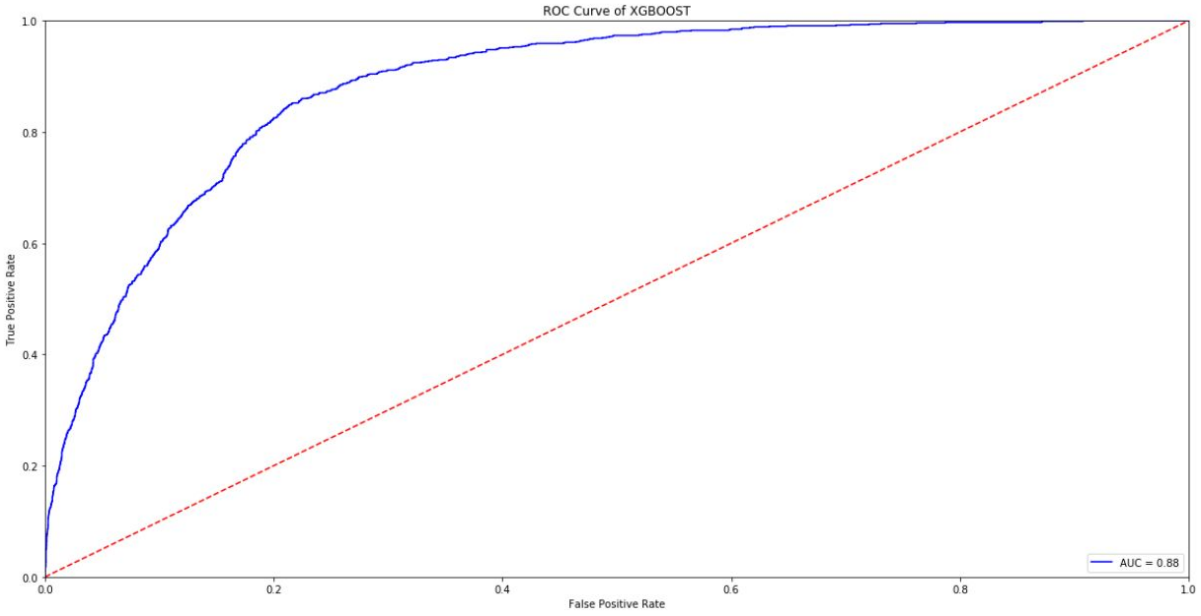
Dando como resultado los siguientes parámetros para la optimización del algoritmo:

```
xgb = XGBClassifier(
 learning_rate =0.01,
 n_estimators=800,
 max_depth=3,
 min_child_weight=7,
 gamma=0.0,
 subsample=0.8,
 colsample_bytree=0.3,
 objective= 'binary:logistic',
 reg_alpha=0.011,
 scale_pos_weight=9)
```

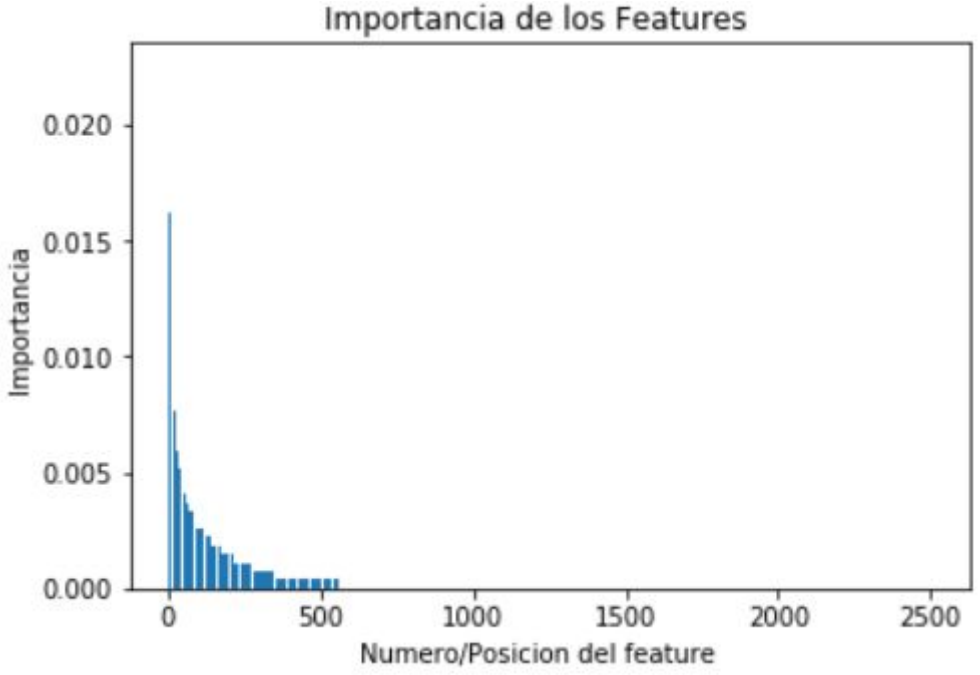
Dando origen a una gran cantidad de árboles similares al siguiente:



Se genera una gran cantidad de árboles simples para realizar nuestras predicciones que con varias herramientas hemos ido evaluando las mejoras producidas, intentando maximizar la métrica AUC.



Antes de entrenar y ajustar el algoritmo se estandarizaron los valores de los features para luego entrenar. Verificamos la importancia de los features sobre este algoritmo, observando que la dimensionalidad de nuestros datos calculada a través de la descomposición en valores singulares, es consistente con la importancia de los features para XGBOOST, tal como se ve en el siguiente gráfico.



Finalmente, luego de culminar con el tuneo de los parámetros, instanciar el modelo, entrenar el mismo, se procedió ha realizar la corrida del algoritmo y su posterior submit en la competencia obteniéndose una puntuación de 0.85154 siendo el mejor resultado obtenido con los algoritmos probados.

4.9. Perceptron

Algoritmo para el aprendizaje supervisado basado en clasificadores binarios. Un clasificador binario es una función que puede decidir si una entrada, representada por un vector de números, pertenece o no a alguna clase específica. Es un tipo de clasificador lineal, es decir, un algoritmo de clasificación que realiza sus predicciones basadas en una función de predicción lineal que combina un conjunto de pesos con el vector de características.

Decidimos darlo de baja instantáneamente porque underfiteaba terriblemente, con un score de 0.58766 en la plataforma de Kaggle. Cabe aclarar que no optimizamos hiperparametros por un tema de que en default XGBOOST ya andaba cerca del 0.80.

5. Conclusión Final

Tal vez la elección de los features fue clave, podríamos haber intentado algo diferente pero tuvimos tiempos acotados, tratando de trabajar el mayor tiempo posible en el TP y de la mejor manera con un integrante menos. Creemos entender que hay alumnos con menos submits que nos superaron por tratar con mejores features ya que desde la parte del algoritmo lo tratamos de la mejor manera posible.

Durante todo el desarrollo del presente trabajo, hemos notado que es fundamental la construcción de features de calidad que abarquen la mayor cantidad de características posible, dado que al trabajar con los algoritmos nos hemos topado con barreras que van más allá de conseguir los hiperparametros adecuados. Por lo que hemos tenido que utilizar técnicas de muestreo, rediseñar features, pero aún así, entendemos que desarrollando mejores features nos sería posible avanzar hacia una mejor solución.,

El algoritmo con el que obtuvimos mejores resultados fue *XGBOOST*, con el que obtuvimos un resultado de ***0.85154,*** siendo un algoritmo robusto, con gran cantidad de hiperparametros, lo cual nos permitió trabajar sobre los mismos, encontrando mejores resultados para nuestro set de datos.

La idea de este informe era la de mostrar el procesamiento de los datos que se hizo para alcanzar el score que obtuvimos finalmente y los diversos algoritmos que se utilizaron a nuestro criterio, ya sea porque pensabamos que ibamos a tener un buen resultado o porque queríamos ver cómo se comportaba nuestro set de datos con otro tipo de algoritmos.