Organización de Datos
-75.06-

Integrantes

Ferreyra Iañez, Javier Hernán, *Padrón Nro. 100680*

Fandos Nicolás Gabriel, *Padrón Nro. 101018*

Scakosky, Matías Ezequiel, *Padrón Nro. 99627*

Grupo 34

2do. Cuatrimestre de 2018
Trabajo practico N°2: Machine Learning
Docente: Luis Argerich

– Lunes-Jueves 19-22 hs
Facultad de Ingeniería, Universidad de Buenos Aires

6 de abril de 2021

Índice

1. Introducción **2**

2. Análisis del set de datos de Trocafone **3**
2.1. Algunas conclusiones del set 3

3. Pre-procesamiento de los datos **4**
3.1. Al principio de todo 4
3.2. Siendo realistas 4
3.3. Un set limpio y listo para trabajar 4

4. Features encontradas **5**
4.1. Features de los usuarios 5
4.1.1. Eventos 5
4.1.2. Visitas 6
4.1.3. Celulares: Precios y modelos 7
4.1.4. Personas que realizaron lo mismo que un usuario 7
4.2. Limpieza del ruido 8

5. Algoritmos probados **9**
5.1. Bagging 9
5.2. KNN 9
5.3. Random Forest 10
5.4. XGBoost 10
5.5. CatBoost 11
5.6. LightGBM 12
5.7. Naive Bayes 12

6. Optimizaciones de algoritmos **13**
6.1. Encodings 13
6.1.1. Mean Encoding (LeaveOneOut) 13
6.1.2. One Hot Encoding 13
6.1.3. Binary Encoding 13
6.2. Stacking 13
6.3. XGBoost+CatBoost 14
6.4. PCA 14
6.5. Feature Selection 14

7. Conclusiones **14**
7.1. La mejor solución 16

1. Introducción

La propuesta de la cátedra para el TP2 de la materia 75.06 Organización de Datos es la `Implementación de una competencia de machine learning`. El objetivo principal de la competencia consiste en realizar pruebas acerca de diferentes algoritmos de aprendizaje supervisado de un set de entrenamiento y luego con un set de test poder contrastar los resultados en la plataforma de Kaggle, para así obtener el mayor score posible.

El informe contiene dos grandes partes, en la primera realizamos un análisis de los datos recibidos de Trocafone y su Pre-procesamiento, el cual luego nos permite sacar features acerca de los usuarios y en la segunda parte del informe, con las features obtenidas y explicadas en la primera parte del informe se exponen los algoritmos de machine learning utilizados para predecir la probabilidad de que un usuario de Trocafone compre o no un celular en la plataforma de e-commerce. Luego de la explicación y desarrollo del algoritmo se explicarán los resultados obtenidos, con estos podremos ir entendiendo que algoritmo nos resultó mejor a la hora de predecir y exponerlo como el que nos resultó mas óptimo

También llegando al final del informe aparece una sección dedicada a como mejorar los algoritmos explicados anteriormente, como mediante el uso de búsquedas de hiper parámetros o pruebas de diferentes tipos de encodings ayudaron a los algoritmos a mejorar su performance. También la combinación de los algoritmos es parte de esta sección, como decidimos combinarlos para buscar que las predicciones de dos o mas algoritmos en conjunto puedan generar una mejor solución, lógicamente los resultados de estas pruebas de combinación también son expuestas al final de este informe

En la ultima sección del informe se encuentran los resultados finales y las conclusiones acerca de todo lo mencionado en este informe, sobre que cosas nos funcionaron más o menos, nuestra posición final en el score de la competencia de Kaggle y alguna opinión nuestra de por qué llegamos a donde llegamos con los algoritmos ejecutados

2. Análisis del set de datos de Trocafone

Esta secciona se ocupa de repasar algunos aspectos que consideramos importantes para el análisis de de los datos, para información mas detallada acerca del análisis exploratorio de los datos del set, está el informe del primer trabajo práctico. Es necesario hacer la aclaración de que encontramos pequeñas diferencias entre el análisis exploratorio del set de datos del TP1 respecto de los datos que obtuvimos.

2.1. Algunas conclusiones del set

Las siguientes conclusiones del set de datos fueron el puntapié inicial para empezar a pensar features de los clientes de Trocafone, si bien no son todas algunas de estas fueron disparadores de ideas de como empezar a pensar características de los usuarios que después serán parte integral del entrenamiento de los algoritmos de aprendizaje

- El celular mas comprado sigue siendo el Samsung Galaxy J5
- El evento mas realizado es Viewed Product
- La mayoría de los usuarios realizan entre 4 y 6 acciones en la pagina
- Continúan siendo bajas las conversiones en la pagina respecto del total de visitas.
- Los iPhone lideran la tabla de los mas vistos, pero no los más comprados
- Según la relación entre las visitas a celulares y las compras, la tendencia pareciera ser querer comprar un iPhone y terminar comprando uno de menor precio
- La región de la de los usuarios es Sao Pablo, seguido de Mina Gerais
- El sistema operativo preferido de las personas del set es Windows 7

3. Pre-procesamiento de los datos

3.1. Al principio de todo

Al empezar el trabajo practico cometimos el error de querer empezar a probar los algoritmos de machine learning sin hacer un pre procesamiento importante de los datos, gracias a esto conseguimos empezar a probar los algoritmos pero los resultados al principio no fueron los mejores, ya que rondaron los $0{,}50000 \pm 0{,}10000$ de score en Kaggle. (Los resultados de esos intentos fueron quitados de este informe y nos quedamos a partir de los intentos con datos pre-procesados)

3.2. Siendo realistas

Si bien aun no teníamos muchas features al momento de hacer esto, nos dimos cuenta que era un error garrafal no tener un pre-procesamiento adecuado de nuestro set de datos y comenzamos primero por hacer algunas limpiezas de columnas, primero las columnas que tenían demasiados valores iguales a cero, o sin valor (Nan) fueron removidas del set de datos. Por otro lado comenzamos un análisis un poco mas manual de algunas columnas que antes de convertirse en features pensamos que debían tener algún tipo de análisis. Las columnas con todos valores iguales fueron descartadas, por mas que no fueran "0.º "Nan", esto nos ayudo a pensar que algunos datos de los brindados por la cátedra no eran representativos para los clientes que realizarían una posible conversión. Es decir que algunos de los datos recibidos "molestan" un poco más de lo que ayudan en la predicción

3.3. Un set limpio y listo para trabajar

Si bien esto puede ir a la parte de conclusiones y resultados del trabajo practico nos pareció que no podíamos dejar de aclararlo desde el principio del informe, debido a esta decisión conjunta del grupo de empezar el trabajo de nuevo, si bien no habíamos avanzado demasiado, volvimos al punto inicial del trabajo practico importando solamente el dataset que nos brindaron y comenzando desde cero a limpiar y mejorar la calidad de los datos, de la forma que mencionamos en el punto anterior.

4. Features encontradas

Durante el transcurso de la realización del trabajo práctico fueron surgiendo diferentes ideas de features para probar. La idea fue siempre ir optimizando la cantidad y la calidad de los features a medida que los algoritmos iban subiendo su score, y descartarlos si veíamos que no podíamos subir mas allá de un valor una vez agregados unos ciertos features. A continuación están descriptos los features de los clientes los cuales para cada cliente se genero una columna diferente por característica hallada dentro del set de datos original brindado por la cátedra

4.1. Features de los usuarios

En esta sección del informe se describen los diversos grupos de features a grupos en Eventos, Modelos y precios de los equipos y visitas. Cada una tiene varios features asociados a los grupos. Muchas de las features fueron explicitadas en el informe, no todas debido a que son muchas, y varias de ellas son parecidas en naturaleza entre sí (con datos diferentes pero que surgen de la misma idea), fueron descriptas las que a nuestro criterio aportan mayor información a los algoritmos de ML para predecir.

4.1.1. Eventos

Mayor evento: El evento que mas veces realizo el usuario, con cada evento de mayor frecuencia en el usuario pensado como una categoría, cada uno de los mismos al usarse en algoritmos que no aceptan variables categóricas se les aplico algún enconding como one hot encoding o mean encoding

Cantidad de eventos: Cada usuario realizo una cierta cantidad de eventos dentro de la plataforma y este feature tiene la cuenta de cuantas veces realizo el usuario una acción o varias diferentes

Vio celulares: Una feature dedicada exclusivamente a si el usuario estuvo o no mirando celulares en la pagina

Mismo mayor evento: La cantidad de personas que realizaron el mismo mayor evento que el usuario en cuestión, sirve como una medida de popularidad de la pagina de los usuarios que tienen una cierta acción preferida y cuanta gente también tiene esa misma acción como su mayoritaria

Clickeo ad: Esta feature determina si el usuario llego a la pagina por algún aviso

Eventos por día: Un promedio de la cantidad de eventos que realizo un usuario por la cantidad de días que estuvo en la plataforma

Primer evento: Es otra feature categórica donde el primer evento que realizo el usuario dentro de la pagina es una categoría para luego ser codificada

Cada evento como una feature: Se nos ocurrió que podía ser una buena idea pensar a cada evento como una feature del usuario y el valor asociado es si realizo ese evento o no, para todos los eventos del data set. Cada feature precisa como True si el usuario hizo el evento o False si no. De esta manera se puede pensar al vector de eventos como 1 y 0 y de esta manera tener una forma mas

precisa de evaluar los eventos para cada usuario mas allá de las cantidades de veces que el usuario realizo cierta acción, además de estar agregando para cada usuario varios, pero simples features

Cantidad de veces que realizo cada evento: De forma análoga al ítem anterior, cada evento representa una feature, pero esta vez, el valor no es de forma booleana si realizo o no el evento, sino tener idea de cuantas veces realizo cada posible acción en la pagina y de esta manera ser mas precisos para construir la predicción del usuario en cuestión

Compras: Cantidad de compras realizadas por el usuario

Vistas a productos: La cantidad de visitas a productos que realizo un usuario fue considerado como una feature importante para los usuarios

4.1.2. Visitas

Dispositivo: El usuario realizó visitas a la página desde diferentes dispositivos, el preferido del usuario fue una feature interesante que encontramos dentro de las visitas a la plataforma.

Cantidad: Cantidad de visitas totales a la página

Primera y última visita: La fecha y el mes de la primera y la ultima visita del usuario son dos features categóricas diferentes a encodear para algunos algoritmos.

Permanencia: La diferencia de días entre la ultima vez que el usuario visitó la pagina y la primera vez.

Hace cuanto visitó: La ultima vez que el usuario entro a la pagina respecto de la última fecha registrada.

Channel: El canal por el que más veces el usuario entró a la página, de forma categórica.

Región: El usuario pudo haber ingresado desde varias regiones diferentes, por lo que la feature que se consideró fue el lugar desde el que ingresó más veces el usuario.

Sistema operativo: El cliente que visitó Trocafone si bien pudo haberlo hecho desde diferentes plataformas y canales, también tiene un sistema operativo asociado el cual nos resultó una feature interesante para tener en cuenta.

Screen size: Obtener el tamaño de la pantalla también nos pareció interesante, ya que esto podría estar relacionado a que la mayoría de las personas prefieren realizar compras desde dispositivos grandes como computadoras o tablets y no desde un celular.

Primer Channel: Referido al canal donde por primera vez el usuario ingresó a la plataforma.

Usó computadora: Ya que se observó en el análisis exploratorio que la mayoría de las compras las realizaron personas que entraron por una computadora, consideramos que podía ser útil ver si la persona ingresó a la pagina por una computadora o no.

4.1.3. Celulares: Precios y modelos

Una cosa que nos pareció interesante dado que conocíamos la página de Trocafone fue investigar y agregar a mano los precios de cada modelo de celular para tener mas datos referentes al costo de los equipos y su relación con las visitas/compras de los usuarios.

Referente a los modelos de celulares tomamos como feature:
La cantidad de modelos distintos vistos: Es una variable numérica útil ya que nos habla de si los usuarios suelen ver varios modelos distintos y esto puede estar relacionado con que los usuarios vayan a realizar una compra.
La marca del celular mas visto por el usuario: Al igual que lo anterior, nos pareció interesante considerar la marca del celular mas visto por el usuario.
Si vio modelos más comprados: Tomamos por separado cada modelo y verificando de forma booleana si vio o no determinado modelo de entre los mas vendidos.
Si vió o compró un modelo con cierto estado: El estado del celular puede ser excelente, muy bueno, bueno o nuevo y en general esta es una característica sobre la que debe decidir el usuario en caso de no querer la opción predeterminada.
Si vió o compró un modelo con cierta capacidad: Al igual que con el estado del celular, esto es una opción que en general el usuario puede elegir y cierta capacidad de memoria interna puede parecerle mas atractiva a los usuarios de forma tal que terminen comprando.

Además con la nueva información de los precios que obtuvimos pudimos analizar para cada usuario:

- El precio del celular que mas vió
- El promedio del precio de los celulares que vió
- El promedio del precio de los celulares que compró
- El precio del celular mas caro que vió
- El precio del celular mas barato que vió
- El promedio del precio del celular más visto en la región del usuario
- El promedio del precio del celular más comprado en la región del usuario
- El promedio del precio de la marca del celular que más vió

4.1.4. Personas que realizaron lo mismo que un usuario

Teniendo en cuenta el efecto que puede producir en los clientes lo que se conoce como efecto cascada y que las recomendaciones que se pueden hacer entre si los usuarios decidimos que podía ser útil tener en cuenta la cantidad de

gente que hacía lo mismo (o que estaba en la misma situación) que un usuario determinado.

Personas con mismo evento más realizado: Una feature interesante era ver cuanta gente realizó más veces el mismo evento que el que más haya realizado un usuario.

Cantidad de personas que comparten características de visita: Decidimos tomar como features individuales la cantidad de personas que entraron desde el mismo dispositivo que un usuario determinado, la cantidad de personas que entraron desde la misma región, las personas que entraron por el mismo canal, las personas que usan el mismo sistema operativo que el que usa un usuario y la cantidad de personas que usan la misma resolución de pantalla que la que mas usa un usuario.

4.2. Limpieza del ruido

Durante buena parte, por no decir casi todo el trabajo práctico estuvimos avocados en encontrar buenas features. También consideramos durante buena parte del TP que mientras mas features consigamos mejor podremos representar a cada usuario, con un vector de mas de 140 dimensiones, de manera óptima. Luego de estancarnos en cierto momento del trabajo practico con nuestras corridas a los algoritmos, nos dimos cuenta que algunas de esas features que agregamos a los usuarios simplemente por agregarlas pensando que mas es mejor, resultaban estar "molestando.ªl algoritmo, de alguna manera el ruido introducido por esas columnas en el DataFrame de features generaban aspectos poco representativos de los usuarios en pos de si iban o no a realizar una conversión, desde un punto de vista algorítmico esto implica iteraciones desperdiciadas en features que no valen la pena revisar, hojas en los arboles en las cuales al analizar su profundidad, se caía en redundancias, por ejemplo en RandomForest, o también se puede traducir a dimensiones en KNN que no solo hacen que una feature no ayude a saber si un usuario compró un producto o no, sino que también genera consumo de recursos del algoritmo haciendo que el mismo aumente su orden de complejidad, su consumo de memoria y el tiempo que tarde en resolver una consulta.

Lo que hicimos para solucionar este problema fue simplemente tomar el dataFrame de features y analizar las features una a una, y pensar de que manera estas podían ser utilizadas para predecir si un usuario compra o no. El criterio fue sencillo: Si para nosotros una característica del usuario puede ser un factor representativo de una posible compra la dejamos en el set, sino simplemente dropeabamos la columna independientemente del tipo de dato que tenga la misma. Un ejemplo de esto podría ser: Analizando la feature Çantidad de compras"donde el valor asociado para cada usuario es la cantidad de conversiones ya realizadas por el mismo, llegamos a la conclusión de que representa al usuario para saber si el mismo puede llegar a querer volver a comprar un producto, si compró ya alguna vez, o varias veces, es posible que el mismo tenga confianza sobre la plataforma para querer realizar otra compra del producto. Por otro lado la feature .ªsk"que indica si el usuario llego a Trocafone a través del motor de

búsqueda de Ask consideramos que no tiene nada que ver con la intención de un usuario de realizar una compra, porque pensándolo bien ¿Que importa que motor de búsqueda use el usuario? Si al fin y al cabo termino visitando nuestra página.

Entonces todo este análisis nos llevo en un principio a pensar sets mas reducidos e ir probando cuanto podíamos reducir las features sin afectar los resultados que teníamos. Por eso fue que intentamos hacer combinaciones que pasaban de tener 140 features a 120, luego a 100 e incluso llegamos a probar bajar hasta 53, donde claramente los algoritmos empezaron a bajar su performance, luego encontramos entre 103 y 118 nuestra mejor relación cantidad/calidad de los features.

5. Algoritmos probados

5.1. Bagging

El bagging consiste en aplicar el mismo clasificador n veces usando Bootstraping (Tomamos muestras del set de entrenamiento (con reemplazo de igual tamaño que este)).

Fue el primer algoritmo que probamos inicialmente cuando teníamos apenas 11 features y no nos presento buenos resultados pues el score que obtuvimos fue de 0.63944. Luego cuando nos acercamos al final de la competición decidimos darle otra oportunidad aprovechando que ahora contábamos con una cantidad muy superior de features (en total 140) y combinarlo con Mean Encoding, sin embargo los resultados siguieron sin ser los mejores así que lo descartamos. Nuestro mayor puntaje usando bagging fue de 0.78614

5.2. KNN

El algoritmo denominado KNN (K-Nearest-Neighbors) se basa en encontrar para un determinado punto sus K-vecinos mas cercanos y luego predecir su probabilidad de pertenecer a una clase, evaluando cuantos vecinos pertenecen a la clase que queremos clasificar y luego dividiendo por el total de los K vecinos.

Para este algoritmo se utilizó Mean Encoding ya que One Hot Encoding generaba muchas columnas lo que hacia muy lento al algoritmo, y luego se normalizaron y estandarizaron las columnas. Lo probamos inicialmente cuando teníamos pocas features y obtuvimos un score de 0.59471, pero luego al obtener mas features y volver a darle una oportunidad vimos que no era muy viable ya que el coste del algoritmo es de orden cuadrático y al querer ejecutarlo varias veces usando GridSearch para hallar los hiper parámetros óptimos el tiempo que tardaba en hacer todas las comparaciones era excesivo.

Decidimos no darlo por descartado debido a que KNN es un algoritmo que se sabe popular y que trae buenos resultados. Por ende decidimos darle una oportunidad, por mas que tengamos una buena cantidad de features. Decidimos para evitar que el algoritmo tarde tanto, probar, basándonos en el soporte

teórico de la materia, una descomposición en valores singulares de la matriz de features y quedarnos con ciertas columnas, con un algoritmo PCA y recién aquí encontramos buenos resultados con KNN. Pasamos de tener un score de aproximadamente 0.6 a 0.83 sin hacer ningún tunning de hiper parámetros. Por lo que decidimos aplicar una PCA que implica una reducción de dimensiones de 118 columnas a 90, y con esto hacer una GridSearch. Lamentablemente luego de unas 4 horas que tardo en realizarse la búsqueda de hiper parámetros de KNN solo conseguimos un score de aproximadamente 0.85 por lo que si bien, pudimos mejorar la predicción de KNN, esta lejos de las primeras posiciones de la competencia y de nuestro máximo score.

5.3. Random Forest

Va a generar varios arboles (la cantidad la determina un hiperparámetro) y cada uno de estos árboles solo va a utilizar un subconjunto de las features que tenga el set de datos.

Usa subconjuntos de arboles de decisión donde cada uno usa un Bootstrap del set de entrenamiento y un cierto conjunto de features tomadas al azar.

Este fue uno de los primeros algoritmos que utilizamos para encarar el problema combinando el uso de mean encoding y binary encoding pero no nos dió los mejores resultados, pues el score que obtuvimos al utilizarlo fue de 0.62012. Como no superaba a los algoritmos de gradient boosting decidimos abandonarlo.

Lo que nos llamo poderosamente la atención de Random Forest es que dadas todas nuestras combinaciones posibles de features que probamos a lo largo de todo el trabajo practico tuvimos curiosos casos de overfitting, ya que cuando dividimos nuestro set de training en un mini set de train y otro mini set de test que nos de una idea de como estaba ajustando nuestro algoritmo a los resultados, al contrastar los resultados con estos dos set pequeños ya notamos que empezábamos a tener un problema gigante de overfitting con este algoritmo, ya que las predicciones sobre el set de entrenamiento rondaban las 0.99797 y las del set de test 0.81514 (i.e. Resultados medidos con métricas internas en nuestros notebooks utilizando ROC AUC)

Este problema nos dio un indicio de que no era buena idea subir los resultados a la página con este algoritmo, ya que probablemente tengamos además de un bajo score, un gran problema de overfitting que no quisimos padecer al final, cuando se abran todos los datos del set de test final. Sabíamos que capaz podíamos obtener un buen score (Con muy baja probabilidad) pero al fin y al cabo eso iba a ser una mentira que tarde o temprano se iba a desenmascarar cuando se contraste contra todo el set.

5.4. XGBoost

Es un algoritmo que intenta hacer mínima una función objetivo (que es una función que mide el error) eligiendo los parámetros que componen la suma de: El error del modelo y el factor de regularización (que mide la complejidad del modelo (si es muy complejo seguramente haya overfitting), de forma tal que se

logra un algoritmo que se ajusta bien a los datos pero que no sea excesivamente complejo tal que overfitee los datos. El error del modelo es la sumatoria de los errores en cada predicción.

La predicción de este algoritmo de boosting es la sumatoria de la predicción de varios arboles y cada árbol nuevo intenta corregir los errores del anterior.

Este algoritmo fue uno de los primeros que probamos ya que conocíamos su potencial. Primero intentamos usar el algoritmo con hiper parámetros puestos por nosotros a mano, para probar sus resultados, luego intentamos hacer una GridSearch y una RandomizedSearch para la búsqueda de estos parámetros. Los resultados de XGBoost a priori resultaron buenos, es decir, cuando tuvimos features suficientes como para considerar que el algoritmo iba a funcionar bien comenzó disparando nuestro puntaje, y si bien es el algoritmo que mas oportunidades le dimos a lo largo de todo el trabajo practico no pudimos hacer que nos de un mejor score por sí solo.

Consideramos varios encodings diferentes para usar XGBoost pero lo que pareció ser el algoritmo que iba a disparar nuestro score, terminó quedándose en una simple promesa, y no entendimos bien si es por que nosotros no lo supimos aprovechar o porque no era la solución óptima para este problema. El ultimo resultado que nos dio XGboost fue un score de 0.85959

5.5. CatBoost

Este es un algoritmo de gradient boosting que, a diferencia de XGBoost, acepta variables categóricas (De ahí su nombre, que es una combinación de las palabras “Category” y “Boosting”)

Este algoritmo es uno de los más lentos, así que para hallar sus hiperparámetros se fue probando a mano leyendo la documentación ya que un Grid Search hubiese tomado demasiado tiempo para probar todas las combinaciones.

Aunque generalmente este algoritmo no suele dar los mejores resultados, sorprendentemente fue el que nos dió el puntaje más alto de todos por si mismo, con un score de 0.86602. Nos pareció raro que CatBoost nos de un mejor resultado que XGBoost en un principio, ya que es sabido que si este algoritmo da mejores resultados que XGBoost, de alguna manera estamos manejando mal nuestras features categóricas, es decir que podríamos no estar encontrando el mejor encoding o la mejor manera de trabajar con estas features. CatBoost nos ofreció la posibilidad de romper por primera vez la barrera del 0.86 por lo que insistimos con su uso hasta que nos dimos cuenta de las limitaciones que posee este algoritmo, intentamos utilizar ensambles que lo contengan, y eso esta detallado en la sección de optimización de algoritmos de este informe, pero la realidad es que somos conscientes que la solución no venía de la mano de CatBoost, así que antes de seguir intentando probar algo que nos daba resultados en el corto plazo, abandonamos su uso y decidimos usarlo de forma tal que nos sirva como punto de referencia para el resto de los algoritmos, el algoritmo o ensamble que supere el 0.86602 que nos dio CatBoost sería nuestro algoritmo a tener en cuenta como próxima referencia.

5.6. LightGBM

LightGBM es un algoritmo de gradient boosting que usa algoritmos de aprendizaje basados en árboles. Esta diseñado para trabajar con variables categóricas y ser mas eficiente que otros algoritmos de gradient boosting ya que la velocidad de entrenamiento es mayor, utiliza menos memoria, tiene mayor precisión, soporta aprendizaje en paralelo y además es capaz de manejar grandes cantidades de datos.

En nuestro caso no funcionó mal, el problema es que nunca alcanzo las expectativas que teníamos con este algoritmo, pensamos que podía llegar a dar mejore resultados, probamos a lo largo de de todo el TP intentar siempre darle una nueva oportunidad a LightGBM pero nunca supero a ninguno de los otros algoritmos, si bien sus resultados no parecían malos en un principio, ya que los mismos oscilaban por el 0.84519, entendemos que los puntajes son relativos a los problemas y un score de no menos de 0.86 nos resulta razonable como para seguir insistiendo con un algoritmo. Por eso fue dado de baja luego de los primeros 3 intentos que no trajo ningún beneficio.

Sin embargo llegando al ultimo día del trabajo práctico cuando los archivos de features habían pasado por diferentes cambios, los cuales incluyen la modificación de algunos features, el agregado de nuevos e incluso la eliminación de características que nos resultaron ruidosas, volvimos a pensar en que LightGBM merecía la oportunidad de correr con los features mejorados en su ultima versión, sin embargo los resultados llamativamente fueron muy parecidos a los de las primeras veces que probamos el algoritmo, es decir, que por mas que hicimos varios cambios en el las features y nuestro set de características de los clientes en general, los resultados del algoritmo fueron muy parecidos tanto al principio como al final del trabajo practico

5.7. Naive Bayes

Es un algoritmo extremadamente sencillo, rápido y que escala muy bien basado en el teorema de Bayes. Casi nunca es 'el mejor' algoritmo pero tampoco suele ser el peor en la gran mayoría de los problemas. Sabíamos de antemano que este algoritmo no nos iba a brindar grandes resultados, pero llegado un momento de estancamiento con el resto de los algoritmos decidimos probar Naive Bayes solo porque teníamos tiempo para ver que tan lejos llegaba acercándonos relativamente al final de nuestro trabajo, con el set de features mas avanzado que logramos crear. El algoritmo no resultó para nada bueno, como nos imaginamos de antemano, los resultados obtenidos resultan inadmisibles para los que se manejan en las tablas de Kaggle, entendiendo que al obtener un score de 0.73484 nos despostaría en las últimas posiciones de la tabla, y ya en este momento habiendo obtenido nuestro mayor score posible en la competencia, antes de seguir considerando este algoritmo, decidimos dejarlo como algo anecdótico, porque ya que estuvimos probando varios algoritmos, este no deja de ser una opción entre los mismos. Puede que en otro tipo de problema de buenos resultados, pero al obtener el puntaje que nos dejó la primera vez que lo corrimos

decidimos abandonarlo y no volver a abrir el notebook correspondiente a sus test.

6. Optimizaciones de algoritmos

6.1. Encodings

6.1.1. Mean Encoding (LeaveOneOut)

El primer encoding que se decidió utilizar fue Mean Encoding, el cual se trata de codificar cada categoría como el promedio del Target, considerando sólo las filas que contienen esa categoría. LeaveOneOut significa que calcula ese promedio dejando de lado la fila que está codificando, de ésta forma se evita un filtrado grosero de los labels en el set de entrenamiento.

Para probar su funcionamiento se dividió el set de entrenamiento en sets de train y test, el set de train se codificó con los labels y el de test sin ellos. Los resultados del score interno fueron similares a cuando utilizamos los labels para codificar todo el set y luego dividirlo.

Este encoding fue el que dió mejores resultados.

6.1.2. One Hot Encoding

Otra forma de encoding que se probó fue One Hot Encoding, la cual genera una nueva columna por cada categoría con unos o ceros según si la muestra pertenece a la misma.

Esta codificación no nos dió mejores resultados por lo que fue descartada.

6.1.3. Binary Encoding

Ésta forma de encoding asigna un numero entero a cada categoría y lo convierte a binario de n dígitos. Luego genera n columnas donde cada una es uno de esos dígitos.

Con este encoding se consiguieron resultados similares a los de One Hot Encoding por lo que fue descartado.

6.2. Stacking

Stacking es una forma de combinar múltiples modelos ya que la idea del stacking es separar el set de entrenamiento en varios sets disjuntos, entrenar varios algoritmos con una parte y utilizarlos para predecir el resto con Cross-Validation, luego utilizar estas predicciones obtenidas como features para otro clasificador. La idea es que este ultimo aprenda a ponderar los primeros para utilizar las mejores predicciones de cada uno.

En nuestro caso combinamos primero XGBoost, CatBoost y LightGBM. El resultado fue favorable ya que se obtuvo una mejora con respecto a los mismos algoritmos por separado y nos dió el score más alto hasta ese momento. Luego se agregó KNN y Random Forest pero no hubo mejora notable.

6.3. XGBoost+CatBoost

Esta combinación fue la que nos dió nuestro mejor score en Kaggle. En este caso lo que hicimos fue ponderar las predicciones de los dos algoritmos que nos dieron mejores resultados por separado, fuimos probando distintas combinaciones. La combinación que mejor resultado nos dió fue $(0{,}1 \cdot xgboost + 0{,}9 \cdot catboost)$ cuando teníamos aproximadamente 133 features, el cual nos dio en Kaggle un score de 0.86636.

6.4. PCA

PCA es una técnica que puede ser útil para hallar las features mas importantes de un dataset, consiste en proyectar los puntos en nuevos ejes de menor dimensión que expresen de la mejor forma la variabilidad de los datos. El resultado es equivalente a utilizar la aproximación que da la SVD con las columnas normalizadas.

Esta descomposición nos fue muy util a la hora de usar KNN, como ya se mencionó en la sección dedicada a este algoritmo.

6.5. Feature Selection

A medida que fuimos avanzando en el trabajo, agregamos nuevas features buscando encontrar algunos que mejoren las predicciones, el primer salto ocurrió al pasar de tener aproximadamente 15 features a tener 35, el score en Kaggle pasó de 0.74 a 0.85.

Luego de seguir agregando features no se notó una mejora hasta llegar a aproximadamente 80, cuando hubo un salto en el score de 0.855 a 0.865.

Al pasar las 80 features en nuestro set de datos observamos que las predicciones no mejoraban, así llegamos a tener aproximadamente 150 features sin observar ninguna mejora notable (el score máximo fue de 0.86636), por lo que decidimos limpiar el set de entrenamiento buscando aquellas features menos significativas o que incluso podrían ser ruido.

Luego de remover una gran cantidad de features nos quedamos con un set de datos con 113 features. Luego de probar esto con los algoritmos que nos habían dado mejores resultados no notamos una mejora.

7. Conclusiones

Llegando al final de nuestro trabajo practico comenzamos a formar nuestras propias conclusiones relacionadas a los algoritmos explicitados en el transcurso del informe, y en relación, también, a la ciencia de datos en general explicada durante todo el cuatrimestre en la materia.

Para empezar, nos dimos cuenta de lo que a priori nos pareció obvio, ya que los docentes del curso se encargaron de hacernos entender. Primero, la importancia de conseguir buenos features para generar buenas predicciones, esta es

un arma importantísima para tener un algoritmo dotado de herramientas suficientes para poder generar una predicción buena, es decir que si alguien quiere hacer una predicción mediante algoritmos de ML y no se gasta en conseguir buenas features, o la mayoría son simplemente ruido, por mas sofisticado que sea el algoritmo, por mas que los ensambles sean una combinación fabulosa del aprendizaje supervisado y los hiper parámetros sean buscados con otro algoritmo de búsqueda exhaustiva, no va a llegar a ningún resultado interesante, es mas, podríamos afirmar que otra persona que encontró mejores features, con llegar a probar un algoritmo solo, sin ninguna optimización, por ejemplo KNN normalizado podría superarlo, de aquí la importancia de buscar buenas features.

En nuestro caso nuestros mayores saltos en precisión y score se dieron gracias a la obtención de features, y luego las técnicas antes mencionadas fueron utilizadas para optimizar estas predicciones.

Por otro lado, lo que nos han advertido, y nos encontramos también durante el proceso de realización del trabajo práctico fue el problema del sobre ajuste (Overfitting) que tuvimos intención de corregir mientras hacíamos las pruebas de los sets de train y test y creemos que llegamos buenos resultados y algoritmos que no sobreajustan a los datos de entrada, mediante la modificación de hiper parámetros y ajuste de las features, como fue explicado anteriormente.

Respecto a KNN pudimos apreciar que el algoritmo funciona bien para pocas dimensiones, pero a medida que fue se fueron creando features nuevas para los usuarios y las mismas se consideran una nueva dimensión para el algoritmo, KNN comienza dejar de ser una opción viable, debido a que comienza a tardar muchísimo en predecir a raíz de un set de test. Mucho mas si hablamos de hacer una búsqueda de hiper parámetros para el algoritmo, probando diferentes métricas o cambiando la cantidad de vecinos cercanos que queremos tomar, como mencionamos antes, esa búsqueda tardo, en una PC con buena cantidad de memoria y procesamiento, al rededor de 4 horas y 15 minutos, lo cual resulta excesivo en comparación a otros algoritmos considerando que el set de datos no superaba las 40000 muestras.

Acercándonos a la solución óptima dentro de nuestro TP, tenemos que mencionar a los algoritmos de gradient boosting, los cuales son los que luego son los que nos harían llegar al techo de nuestro score, brindándonos lo que para nosotros fue la mejor solución al problema de machine learning, lo que pudimos concluir acerca de estos algoritmos es que son buenos pero necesitas una buena refinación de sus hiperparámetros debido a que la varianza de los resultados que ofrecen entre una búsqueda por fuerza bruta de los mismos, comparado con una búsqueda greedy suele ser muy grande, y en problemas como estos necesitamos la mejor combinación de hiper parámetros para llegar a una buena solución, estos algoritmos de boosting nos la ofrecieron luego de varios intentos y diversas pruebas y búsquedas de sus hiper parámetros.

7.1. La mejor solución

Como ya anticipamos antes, la mejor solución se dio a partir de la combinación de dos algoritmos de gradient boosting, siendo CatBoost y XGBoost los algoritmos elegidos para la predicción. La predicción final se realizo haciendo un promedio ponderado entre las probabilidades obtenidas por cada algoritmo de la siguiente manera.

$$P(Conversion) = P(CatBoost) \cdot 0{,}9 + P(XGBoost) \cdot 0{,}1$$

Respecto de CatBoost, nos resultó interesante debido a que pudimos utilizarlo para hacer predicciones de forma rápido y sin tanta necesidad de procesar los datos, es decir, las features no necesitan ser codificadas para que el algoritmo pueda predecir ya que como mencionamos anteriormente, acepta variables categóricas y además acepta variables numéricas.

También la ponderación incluye a XGBoost que si bien por si solo el resultado de ese algoritmo no es el mejor, de alguna manera mejora al CatBoost.

Encontrada esta como la mejor solución no fue suficiente para llegar a los primeros puestos de la competencia, por alguna razón que desconocemos y aun estamos intentando entender, incluso en la redacción del final del informe, es por que no pudimos romper la barrera del 0.87, con todas las cosas que probamos, si bien el resultado nosotros lo consideramos satisfactorio, entendemos que pudimos habernos equivocado en la creación de features o en la manera que probamos a los algoritmos.