FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico Nº 2

Machine Learning

Organización de Datos

Nombre grupo: "The beauty and the data"
Nº grupo: 38
Todo el trabajo realizado puede encontrarse en el siguiente repositorio de github:
https://github.com/sebalogue/tp2-datos

Participantes	Nº padrón	Mail
LOIS,Lucas Edgardo	98923	lucaslois95@gmail.com
LOGUERCIO, Sebastian Ismael	100517	seba21log@gmail.com
MARIANI, Santiago Tomás	100516	santiagomariani2@gmail.com
MARIJUAN, Magalí	100070	maguimar001@gmail.com

Índice

1. Introducción

El objetivo de este trabajo práctico es diseñar un Modelo de Machine Learning para predecir la probabilidad de que un cliente compre. En el siguiente informe describiremos todas las etapas por las que pasamos hasta llegar a un modelo predictor de aproximadamente 86 % de precisión.

2. Feature Engineering

La primera etapa de un modelo de Machine Learning (de ahora en más ML) es obtener, en función de los datos, características de cada usuario que luego usaremos para predecir (de ahora en más features).

Para obtener los features se tuvo en cuenta la información obtenida del análisis exploratorio realizado en el Trabajo Práctico anterior.

2.1. Descripción general del set de datos

Antes de comenzar a enumerar los features es imprescindible describir el set de datos en el cual trabajamos.

El set de datos cuenta con eventos que realizaron los usuarios que ingresaron al sitio hasta el 31/05/18. Además, contamos con los labels de una parte de los usuarios que compraron un producto, y lo hicieron entre el 1 y el 15 de junio, por lo tanto se trata de un problema de aprendizaje supervisado. En dicho problema tenemos que predecir, para el resto de los usuarios que no tienen labels, si hicieron una compra o no en el periodo antes mencionado (1 a 15 de junio). Es decir, se trata de un problema de clasificación binaria, indicando con un 1 si el usuario compra o con un 0 si no lo hizo.

Una vez que arrancamos a buscar features nos encontramos con diversos problemas. Uno de ellos fue que para algunos features era necesario filtrar por algún evento y faltaba información.

2.2. Eventos

Manejo de features categóricos

Antes de continuar con la descripción de los features vale aclarar que los algoritmos que usamos no pueden reconocer variables categóricas, solo numéricas. Para solventar esto hicimos un encoding de las variables categóricas. Usamos la técnica de Óne hot encoding'que nos permite pasar de una columna categórica a muchas columnas numéricas donde cada una de ellas se corresponde a un valor de la variable categórica. En cada fila, cada columna es 1 o 0 según si la variable categórica toma el valor de la columna. En algunos casos se realizó algo muy similar partiendo del Óne hot encoding'. Por ejemplo, en el dataframe tenemos información sobre los modelos de celular que visita cada usuario. Entonces, como feature numérico se nos ocurrió obtener la cantidad de visitas de los usuarios a cada modelo (Es decir, cantidad de visitas al modelo Iphone, cantidad de visitas al modelo Samnsung G7, etc). De esta manera, creímos que se podía resumir bastante información en un mismo feature.

Manejo de valores nulos

También es importante mencionar que en algunos de los features mencionados a continuación, ocurre que al filtrar por un evento determinado se pierden personas de las cuales no existe cierta información sobre dicho evento. Estas personas luego hay que agregarlas y por lo tanto aparecen valores nulos. Es aquí cuando hay que pensar con que valor se puede completar esos valores nulos.

En algunos casos (por ej. cuando filtramos por viewed product, conversion, checkout, etc.) se pueden completar con 0s debido a que significa que la persona no realizo el evento. En otros casos los valores nulos pueden significar información faltante (por ej. región desde la cual la persona entra). En estos casos hay varias alternativas. Cuando se trata de features categóricos el valor nulo se puede remplazar por la moda. En el caso de features numéricos lo que se hace es poner la mediana (o el promedio, pero hay que tener cuidado con los outliers[1]).
Otra forma mas compleja es utilizar un modelo para predecir qué poner en los valores nulos.

Cantidad de veces que el usuario hace cada evento

Lo más básico que se nos ocurrió fue determinar la cantidad de veces que cada usuario realiza un evento en particular, es decir, cuántas veces hace una compra, un checkout, una entrada al sitio por campaña publicitaria, etc. Este feature fue el que mayor precisión nos dio por sí solo a la hora de predecir. Si probamos entrenar el modelo con cada feature individualmente, éste es el que mejor puntaje arroja.

Luego hicimos otro feature idéntico pero ahora solo considerando la cantidad de veces que la persona realiza un evento en el mes 5. Este feature nos dio incluso una mejora en el puntaje. Esto creemos que se debe a la cercanía con el periodo del cual debemos obtener la predicción.

Cantidad de modelos distintos vistos por cada usuario

Este feature representa la cantidad de modelos distintos que una persona visitó alguna vez durante el periodo con el que trabajamos. Creímos que este feature podía llegar a aportar información ya que pensamos que si una persona visita siempre el mismo tipo de producto, podría deberse a que ya tenga decidido comprarlo.

Cantidad de visitas de c/persona a distintos modelos

Para este feature necesitamos seleccionar una cantidad determinada de modelos, debido a que estos son muchos e implicaría tener muchas columnas y demasiados ceros, algo que notamos que empeora la predicción del algoritmo. Con esto en mente, seleccionamos solamente los 15 modelos que mas información aporten (una alternativa es tomar los top 15 mas populares, mientras que otra opción que probamos es analizar el set de eventos con labels para buscar anomalías o diferencias grandes entre los que compran y los que no) y, a los usuarios que no visitaron ninguno de estos 15 modelos, simplemente le asignamos ceros a su fila correspondiente.

Persona visita la página con una resolución dada

Para este feature calculamos la cantidad de veces que entra cada usuario con cada resolución. Aquí también tuvimos que seleccionar una cantidad de resoluciones determinada. Para las personas de las cuales no tenemos información remplazamos los valores nulos con la moda ('360x640').

Cantidad de checkouts de c/persona a distintos modelos

Este es similar al de la cantidad de modelos distintos vistos por cada usuario, solo que ahora el evento que consideramos es el checkout.

Cantidad de visitas a producto por color

Representa cuantas veces una persona visita modelos de un color dado. Como hay un listado muy grande de colores, de nuevo fue necesario tomar los K mas importantes.

[1]Outlier: Valores que se encuentran en los extremos de la distribución y distorsionan el promedio.

Cantidad de visitas a productos por storage

Similar al anterior, solo que nos concentramos en el almacenamiento que elije la persona. Como no son tantos los valores de almacenamiento, optamos por incluimos a todos.

Cantidad de visitas a productos por condición

Similar al anterior, solo que nos concentramos en la condición del producto. De nuevo optamos por incluir todos los valores, ya que son solo 5 condiciones posibles.

Regresa o no la persona

Esta información la obtuvimos del evento 'visited site'. Para las personas que no teníamos información sobre este evento optamos por hacer un análisis y ver cuando entraban al sitio. Si la persona entraba aunque sea dos días distintos entonces considerábamos que la persona volvía al sitio, en caso contrario no lo hace.

Región desde la cual visita la persona

Aquí nuevamente se nos dio el problema de los valores nulos. Como se trata de un feature categórico, remplazamos los valores nulos por la moda. Esto es, a la persona de la cual no tenemos información sobre la región, le asignamos la región mas popular ('Sao Paulo').

Dispositivo desde el cual entra la persona

Los dispositivos desde los cuales puede acceder una persona son: Tablet, Smartphone, Computer o dispositivo desconocido. Aquí también decidimos remplazar los valores nulos por la moda ('Smartphone').

Versión de sistema operativo que usa la persona

Aquí decidimos solo tomar en cuenta las k versiones de SO que mas información nos aportaban, ya que vimos que había varios con una cantidad relativamente poca de personas. Nuevamente reemplazamos los valores nulos con la moda ('Windows 7').

Versión de navegador que usa la persona

Lo manejamos de la misma manera que el feature anterior. En este caso la moda es 'Chrome 66.0'.

Cantidad de visitas desde una campaña publicitaria

Representa la cantidad de entradas al sitio de las personas a través de cada campaña publicitaria. Si una persona no entro nunca desde alguna campaña, simplemente llenamos toda la fila con ceros.

El usuario ingresó por una campaña publicitaria alguna vez

Es distinto al anterior ya que sólo puede contener valores booleanos. Los valores nulos los completamos con False ya que suponemos que si no hay información es porque no ingresó por una campaña publicitaria.

Campaña publicitaria por la que ingresó el usuario

Para elaborar este feature lo que hicimos fue usar one hot encoding para el atributo campaign source del evento Ad Campaign Hit.

Cantidad de visitas por el canal desde el cual se conectaron

Del atributo Visited Site usaremos el atributo channel y atribuiremos por usuario la cantidad de visitas desde cada canal.

Para los usuarios para los que no teníamos información los completamos con la media.

El usuario realizó una visita por un determinado canal

Es muy parecido al anterior pero en vez de usar la cantidad pusimos un valor booleano dependiendo si la cantidad era o no mayor que cero.

Cantidad de veces que buscó una palabra clave

Para este feature analizamos en campo searched term del evento Searched product y lo que hicimos fue analizar si en el término de búsqueda se encontraba alguno de los siguientes strings:

- iphone
- samsung
- galaxy
- lg
- sony
- apple
- note
- moto

Para los aquellos usuarios que no teníamos información sobre este feature lo completamos con la media.

2.3. Temporales

Distancia entre ultima entrada al sitio y el 31/05

Este feature lo consideramos importante ya que creímos que si una persona no tiene registro de haber entrado al sitio hace varios días o meses (con respecto al 31/05), es menos probable que entre a la página luego del 31/05 a comprar un producto. Para crear el feature, calculamos cuantos días hubo desde la ultima conexión de cada persona hasta el día 31 del mes 5.

Promedio de eventos por día

Este feature es simplemente obtener un promedio de la cantidad de veces por día que entra cada usuario a la pagina (teniendo en cuenta solamente los días en que el usuario entra, y no todos los días del año). También nos pareció importante crear otro feature igual pero solo teniendo en cuenta el ultimo mes (mes 5) ya que, como vimos antes, el mes 5 nos puede aportar información clave para las predicciones.

Promedio de eventos por mes

El promedio de entradas por mes de cada usuario nos pareció que podía llegar a ser interesante ya que puede dar una idea de cuantas veces entra el usuario a lo largo del año antes de comprar o no un producto. Lamentablemente no fue un feature que ayudó a aumentar el puntaje de las predicciones.

Mes que entró por primera vez

Este feature se nos ocurrió hacerlo ya que creímos que es menos probable que alguien compre un producto en el mes 6 si su primera entrada al sitio fue en el mes 1.

Distancia en días entre primer y ultimo día de entrada al sitio

Para la distancia en días simplemente calculamos la cantidad de días que hubo entre el primer y el ultimo evento registrado en el sitio de cada persona.

Cantidad de días distintos de entrada al sitio

Este feature se nos ocurrió que, junto con el promedio de entradas por día de cada usuario, podría llegar a darle más información al algoritmo. Simplemente lo que hicimos fue ver cuantos días entró al sitió cada usuario, independientemente de cuantas veces entró cada día.

Entró por primera vez al sitio en el mes 5

Como creímos que el mes 5 era el mes que más información aportaba, decidimos ver cuales eran los usuarios que entraron por primera vez (fueron registrados como ÑNew') en el mes 5.

Cantidad de eventos en un mes

Representa la cantidad de eventos totales que hace una persona en cada mes.

Cantidad de eventos en un día del mes

Representa la cantidad de eventos totales que hace una persona según el numero de día que accedió (1 al 31). Es decir, sin distinguir entre meses distintos.

Cantidad de eventos según quincena

Contamos por un lado con la cantidad de eventos que realiza una persona en la primer quincena, y por otro la segunda quincena. Otra vez, sin distinguir entre meses distintos.

Cantidad de eventos por día de semana

Aquí lo que hicimos fue sumar la cantidad de veces que el usuario realiza un evento por un lado los días lunes, por otro lado los días martes, etc. (poniendo una columna para cada día)

Cantidad de eventos en fines de semana

Es la suma de la cantidad de eventos que hace la persona los sábados y domingos.

Cantidad de eventos en los días martes, miércoles y jueves

Es la suma de la cantidad de eventos que hace la persona dichos días. Esto supusimos que era mejor analizarlo por separado debido a que notamos en el TP1 (Análisis exploratorio del mismo set de datos) una gran diferencia en las visitas con respecto a los otros días de la semana. Esto es, mayor cantidad de visitas.

Cantidad de eventos en un rango horario

Representa la cantidad de eventos que la persona hace un rango determinado de horas.

Hicimos dos features de este estilo. Uno de ellos con un paso de 4 horas, esto es: 0-3 , 4-7 , 8-11 , 12-15 , 16-19 , 20-23. El otro separando por parte del día, esto es: mañana (5 a 12), tarde (12 a 17), tarde-noche (17 a 21) y noche (21 a 4).

Cantidad de eventos en rango de minutos

Representa la suma de eventos que la persona realiza en un rango de minutos que puede ser: 0-19, 20-39, 40-59.

Cantidad de eventos en semana del año

Representa la suma de eventos que la persona realiza en cada semana del año. Aquí también lo que nos sirvió fue seleccionar las top k semanas con más eventos totales.

Cantidad de eventos en día del año

Es la suma de la cantidad de eventos en cada día del año, desde el día 1 (01/01) hasta el 151 (31/05). Poniendo cada día como una columna. Aquí también nos pareció necesario elegir los top k días del año.

Ultima fecha de conversión

Aquí lo que hicimos fue seleccionar el número de día del año (entre 1 y 151) en la cual cada persona realizó una conversión. Las personas que no realizaron ninguna conversión decidimos asignarles un cero

Ultima fecha de checkout

Similar al feature anterior pero seleccionando el ultimo día que realizó un checkout.

Ultima fecha de conexión

Número de día en el cual el usuario ingresó al sitio.

2.4. Lagged features

Cada cuantos días hace eventos

Representa cada cuantos días la persona, en promedio, realiza un evento. También hicimos otro feature similar pero considerando solo las entradas en el mes 5.

Cada cuantas horas hace eventos

Representa la cantidad de horas en promedio que pasan entre eventos consecutivos, considerando todos los meses.

Cada cuantos minutos hace eventos

Es la cantidad de minutos en promedio que pasan entre eventos consecutivos, considerando todos los meses.

Cada cuantas horas hace eventos en un día en el mes 5

Representa un promedio de cada cuantas horas hace un evento la persona cada vez que entra un día cualquiera del mes 5. Para hacer esto, calculamos para cada uno de los días del mes 5 un promedio de cada cuantas horas hace un evento la persona, y luego un promedio final de todos esos días.

Cada cuantos minutos hace eventos en un día y hora del mes 5

Representa en promedio cada cuantos minutos la persona hace eventos cuando entra a una hora y día cualquiera del mes 5. Aquí lo que hicimos fue calcular un promedio de cada cuantos minutos la persona realiza un evento en una hora particular de un día del mes 5, luego se obtiene el promedio para todo el día y finalmente para todos los días del mes.

Cada cuantos segundos hace eventos en un día y hora del mes 5

Este feature es similar al anterior, solo que se trata de un promedio de segundos.

Cada cuantas horas hace eventos en su ultima día de conexión en el mes 5

Este feature es idéntico al de 'Cada cuantas horas hace eventos en el mes 5'pero solo considerando el ultimo día de conexión en este mismo mes.

Cada cuantos minutos hace eventos en su ultimo día de conexión en el mes 5

Este representa un promedio de cada cuantos minutos hace eventos pero solo tomando en cuenta la ultima fecha de conexión del usuario.

Cada cuantos segundos hace eventos en su ultimo día de conexión en el mes 5

Igual al feature anterior solamente que haciendo un promedio sobre los segundos.

Cada cuantos días hace conversiones

Representa un promedio de cada cuantos días la persona realiza una compra teniendo en cuanta todos los meses con los que trabajamos.

Cada cuantos días hace checkout

Representa un promedio de cada cuantos días la persona realiza un checkout teniendo en cuento todos los meses con los que trabajamos.

Cuántas veces se registra un evento en particular por día del año

Representa la cantidad de veces que un usuario registra este evento por día del año. Lo días del año se enumeran del 1 al 152.

Los eventos para los que realizamos este análisis fueron:

- Viewed Product
- Visited Site
- Brand Listing
- Serched Product
- Checkout

2.5. Selección de features

Como llegamos a tener una gran cantidad de features, buscamos la forma de poder seleccionar los que mejor información le aporten a los algoritmos a la hora de entrenar y predecir. En una primera instancia intentamos hacerlo sin ayuda de algoritmos especiales, eligiendo los features que nos parecían más importantes y los íbamos probando con los algoritmos. De esta forma pudimos llegar a un puntaje que rondaba el 85, casi 86 % (con métrica ROC AUC). Luego, investigando, probamos unos algoritmos de selección de features que intentan determinar la mejor cantidad de features posibles, o los k mejores features. Algunos son:

RFE (Recursive Feature Elimination)

Dado un estimador externo que asigna pesos a los features, el objetivo de RFE es seleccionar features recursivamente considerando un conjunto cada vez más pequeño de features. Recibe una parámetro ñ'que es la cantidad de features a seleccionar. Este algoritmo lo utilizamos junto con XGBoost y, haciendo varias pruebas, pudimos aumentar el resultado total, seleccionando una cantidad determinada de features.

RFECV (Recursive Feature Elimination and Cross-Validated selection)

Este algoritmo esta basado en la misma idea que el anterior pero ahora basándose en los puntajes obtenidos a través de un estimador de Cross-Validation. Además puede devolver la cantidad 'óptima'de features, es decir, la cantidad de features con la cual se llegó al máximo score a través de Cross-Validation.

Selección de feaures con Random Forest

Otro forma de obtener una idea de cuales son los features más importantes es a través del puntaje que el Random Forest Classifier le asigna a cada feature al hacer un entrenamiento con un dataset determinado. Intentamos con esto filtrar los features con menos importancia para el random forest pero, aunque tuvimos relativamente un buen puntaje, no pudimos sobrepasar nuestro mejor resultado.

3. Balanceo de datos

El set de datos con el que trabajamos está desbalanceado. Es decir, existe una gran diferencia entre la cantidad de personas que compraron un producto entre el 1/06 y 15/06 y las que no. En lo que se refiere a las personas con labels, el 95 % no compro un producto en el periodo antes mencionado. Esto implica que si para todas las personas predecimos que no van a comprar, la métrica accuracy nos da un 95 %. Esto no es bueno, ya que el numero de falsos negativos (los compradores que son predecidos como no compradores) seria muy grande, de hecho seria máximo. Es por esto, que utilizamos otras métricas, como pueden ser ROC AUC (area bajo la curva ROC), F1 Score, precision, recall, matriz de confusión, etc. La métrica que buscamos mejorar fue la de ROC AUC.

Para resolver este problema de balanceo de datos y no caer en overfitting, probamos distintas alternativas: random oversampling, random undersampling y SMOTE.

3.1. Random Oversampling

La idea de este método es aumentar la cantidad de datos de la clase minoritaria de manera aleatoria. Se elige un número determinado para aumentar la clase minoritaria y luego se repiten filas de esta clase al azar. Este método de balanceo es el que más nos sirvió a la hora de aumentar

la métrica elegida.

3.2. Random Undersampling

Random Undersampling: Este método sigue la misma idea del random oversampling solamente que ahora lo que se busca es reducir la clase minoritaria.

3.3. SMOTE - Synthetic Minority Over-sampling Technique

Este algoritmo crea filas sintéticas que sean de la clase minoritaria. Para esto, elige una determinada cantidad de muestras que mejor representen a esta clase e intenta generar filas parecidas.

4. Búsqueda de hiper-parámetros

Algo que es necesario hacer para poder mejorar las predicciones, es encontrar los mejores hiperparámetros para los algoritmos. Para ello vimos distintas alternativas: GridSearch, RandomSearch y Bayesian optimization.

4.1. GridSearch

Es un algoritmo que puede recibir un diccionario con los hiper-parámetros como clave y como valor los valores que puede tomar cada hiper-parámetro, y luego se hace un cross-validation con cada una de las combinaciones posibles de hiperparametros. Finalmente, basándose en el score de cross-validation, se elijen la mejor combinación. Sirve, ya que probamos muchas combinaciones, pero la desventaja que tuvimos al usarlo es que tarda mucho tiempo en finalizar el proceso de búsqueda.

4.2. RandomSearch

Este algoritmo es muy similar al anterior, la diferencia es que prueba combinaciones de manera random. Nos resulto muy útil ya que encontró muy buenos hiper-parametros rápidamente (fijándonos siempre en la métrica de ROC AUC).

4.3. Bayesian optimization

Es otra forma mas compleja de encontrar los hiper-parametros. Es necesario indicar una cantidad de iteraciones, ya que el algoritmo va a convergiendo a los mejores hiper-parámetros, y llega un punto en el que la métrica no mejora. Este algoritmo esta basado en un modelo probabilístico.

5. Transformaciones

Como los datos del set de entrenamiento al que llegamos tiene valores muy dispersos y cuenta con outliers, es buena idea aplicar una transformación a los datos. En general los algoritmos de M.L. suelen predecir mejor cuando se estandarizan los las muestras. También existen otras transformaciones mas aplicadas para los outliers. Es por eso que probamos varias transformaciones, entre ellas: StandardScaler, RobustScaler, MaxAbsScaler, MinMaxScaler, normalize, etc. Cabe

destacar que para los algoritmos basados en árboles este tipo de transformaciones no son de gran aporte.

6. Modelos

A lo largo del trabajo probamos distintos algoritmos de Machine Learning. Todos los algoritmos que probamos los hicimos con una gran cantidad de features y tratamos de seleccionar los que más aportaban para el entrenamiento y la predicción.

6.1. XGBoost

El algoritmo de XGBoost fue el que más nos sirvió a la hora de predecir, ya que fue el que más alto puntaje dio con la métrica ROC AUC y con el que menos overfitting alcanzamos.

Por otro lado, este algoritmo fue el que mas sensibilidad tenia con los hiperparametros, por lo que nos resultó importante intentar determinar cuál era la combinación de hiperparámetros óptima. Un parámetro que vimos que hacía aumentar la precisión de las predicciones fue 'scale pos weight', que se usa para controlar el balance de los pesos positivos y negativos.

Otra ventaja que vimos en este algoritmo fue la rapidez con la que entrenaba y predecía en comparación a todos los otros algoritmos, incluso con gran cantidad de datos.

Los resultados obtenidos fueron:

- Utilizando RFE:
  - train accuracy: 0.902066834073788
  - test accuracy: 0.8980169971671388
  - Matriz de confusión:

3392	300
96	95

  - Área bajo la curva: 0.8710343008514235

6.2. KNN

Como primera instancia intentamos aplicar este algoritmo, sin más razón que porque era el único que conocíamos. Lo que no tuvimos en cuenta es que KNN sufre de la maldición de la dimensionalidad y que por lo tanto con tantos features era imposible que este algoritmo diera un resultado útil.

Intentamos reducir las dimensiones con PCA y el resultado mejoró pero no tanto como para que fuera la opción más viable.

Los resultados obtenidos fueron:

- Aplico SMOTE:
  - train accuracy: 0.8190039792596165
  - test accuracy: 0.7456230690010298
  - Matriz de confusión:

1374	474
20	74

- Área bajo la curva: 0.8210745371649626

- Aplico Random Under Sampling:
  - train accuracy: 0.7477426636568849
  - test accuracy: 0.7554067971163748
  - Matriz de confusión:

1398	450
25	69

  - Área bajo la curva: 0.8134296997328913

La cantidad de dimensiones utilizadas con PCA fue de 10.

6.3. SVM y The Kernel Trick

Por otro lado, intentamos aplicar el algoritmo SVM y luego de obtener malos resultados (sobretodo overfitting) recurrimos por aplicar The Kernel Trick, más precisamente RBFSampler.

El objetivo de usar RBF es aumentar las dimensiones para que luego SVM pueda buscar la mejor frontera que divida a los datos.
Los resultados obtenidos fueron:

- Aplico SMOTE:
  - train accuracy: 0.7534366333051972
  - test accuracy: 0.6930998970133883
  - Matriz de confusión:

1268	580
16	78

  - Área bajo la curva: 0.8000253292806484

- Aplico Random Under Sampling:
  - train accuracy: 0.755079006772009
  - test accuracy:0.6400617919670443
  - Matriz de confusión:

1164	684
15	79

  - Área bajo la curva: 0.7774131896472323

6.4. MLP Classifier

Este es un algoritmo de muchas capas (multi capa) que implementa Perceptrón. Esta última es una red neuronal que intenta buscar un hiperplano que separe las clases a clasificar.

Los resultados obtenidos fueron:

- Aplico SMOTE:
  - train accuracy: 0.9561377064994574
  - test accuracy: 0.8717816683831102
  - Matriz de confusión:

1658	190
59	35

  - Área bajo la curva: 0.7399200976328636
- Aplico Random Under Sampling:
  - train accuracy: 0.8132054176072234
  - test accuracy:0.7559217301750772
  - Matriz de confusión:

1398	450
24	70

  - Área bajo la curva: 0.841893478861564

6.5. Random forest

El Random Forest Classiffier fue el segundo algoritmo que más probamos con distintos features y diferentes combinaciones de hiperparámetros ya que fue uno de los que mejor resultado obtuvimos. Sin embargo, no pudimos mejorar el score que alcanzamos con XGBoost. Por otro lado, lo utilizamos para intentar seleccionar la mejor combinación de features posibles ya que, como mencionamos anteriormente, Random Forest puede devolver el puntaje que le asigna a cada feature con el cual entrena. También lo que intentamos hacer fue una especie de ensamble con XGBoost haciendo el promedio o una ponderación de las predicciones (al calcular las probabilidades).

Los resultados obtenidos con Random Forest fueron:

- Aplico SMOTE:
  - train accuracy: 0.8848124924635234
  - test accuracy: 0.829557157569516
  - Matriz de confusión:

1550	298
33	61

  - Área bajo la curva: 0.8380250069079856
- Aplico Random Under Sampling:
  - train accuracy: 0.8363431151241535

- test accuracy: 0.7785787847579815
- Matriz de confusión:

1438	410
20	74

- Área bajo la curva: 0.8486978447084831

6.6. Extra tree classifier

Luego de aplicar este algoritmo los resultados fueron:

- Aplico SMOTE:
  - train accuracy: 0.9876401784637646
  - test accuracy: 0.9186405767250257
  - Matriz de confusión:

1755	93
65	29

  - Área bajo la curva: 0.8593073593073592
- Aplico Random Under Sampling:
  - train accuracy: 0.8549661399548533
  - test accuracy: 0.7579814624098867
  - Matriz de confusión:

1397	451
19	75

  - Área bajo la curva: 0.8515704154002026

6.7. Ada Boost

Luego de aplicar este algoritmo los resultados obtenidos fueron:

- Aplico SMOTE:
  - train accuracy:0.9035029542988062
  - test accuracy: 0.870236869207003
  - Matriz de confusión:

1640	208
44	50

  - Área bajo la curva: 0.8593073593073592
- Aplico Random Under Sampling:
  - train accuracy:0.7990970654627539
  - test accuracy: 0.800205973223481

- Matriz de confusión:

1487	361
27	67

- Área bajo la curva: 0.8442249240121581

6.8. Bagging

En esta sección lo que hicimos fue aplicar bagging, para ello usamos: BaggingClassifier.

El objetivo de la técnica de Bagging es aplicar el mismo clasificador n veces y luego promediar sus resultados para obtener el resultado final. El problema es que aplicar el mismo clasificador n veces nos da el mismo resultado. Es por esto que **bagging** siempre se usa en conjunto con **bootstrapping**.

Bootstrapping consiste en tomar una muestra del set de entrenamiento del mismo tamaño pero con reemplazo. Es decir que un dato puede estar varias veces en la muestra.

Entonces, internamente el algoritmo entrena al clasificador con los bootstraps y obtiene n clasificadores distintos. Luego podemos aplicar estos clasificadores al set de datos original y promediar los resultados para obtener el resultado final.

6.8.1. KNN con PCA

Al modelo obtenido en la seccion (6.2) le aplicamos bagging, los reultados obtenidos fueron:

- Aplico SMOTE:
  - train accuracy: 0.8187326661039431
  - test accuracy: 0.743048403707518
  - Matriz de confusión:

1367	481
18	76

  - Área bajo la curva: 0.8234549138804458
- Random Under Sampling:
  - train accuracy: 0.7511286681715575
  - test accuracy: 0.7461380020597322
  - Matriz de confusión:

1379	469
24	70

  - Área bajo la curva: 0.8141723081882657

Conclusión: Si bien los resultados mejoraron, no fue una mejora significativa.

En ambos casos la cantidad de dimensiones utilizadas para KNN fue 10.

6.8.2. MLPClassifier

Aplicamos bagging a la red nueronal que utilizamos en (6.4). Los resultados obtenidos fueron:

- Aplico SMOTE:
  - train accuracy: 0.8137887374894489
  - test accuracy: 0.649330587023687
  - Matriz de confusión:

1182	666
15	79

  - Área bajo la curva: 0.821215575204937
- Random Under Sampling:
  - train accuracy: 0.7409706546275395
  - test accuracy: 0.4886714727085479
  - Matriz de confusión:

856	992
1	93

  - Área bajo la curva: 0.8418186423505573

Conclusion Si bien los resultados mejoraron, podemos ver que el *test acurracy* de el segundo modelo es muy bajo por lo tanto, este modelo tiene Overfitting.

6.9. K-Means

En este modelo lo que intentamos hacer fue transformar nuestro espacio de datos (X) a un espacio de distancias de cluster, con el objetivo de normalizar los datos y mejorar la capacidad de predicción de nuestros algoritmos clasificadores.

6.9.1. SVC

C-Support Vector Classification es un algoritmo de clasificación que intenta mejorar a perceptrón buscando la mejor frontera posible para separar las clases.
Utilizamos The Kernel Trick para este algoritmo y los resultados obtenidos fueron:

- Aplico SMOTE:
  - train accuracy:0.7592246472928976
  - test accuracy:0.7718846549948507
  - Matriz de confusión:

1428	420
23	71

  - Área bajo la curva: 0.8018732154370453
- Random Under Sampling:

- train accuracy: 0.7562076749435666
- test accuracy: 0.7353244078269825
- Matriz de confusión:

1357	491
23	71

- Área bajo la curva: 0.801320576586534

6.9.2. KNN sin PCA

Aplicamos K-means al set de datos y luego KNN y los resultados obenidos fueron:

- Aplico SMOTE:
  - train accuracy:0.8589774508621729
  - test accuracy: 0.7770339855818743
  - Matriz de confusión:

1448	400
33	61

  - Área bajo la curva: 0.7917501381597126
- Random Under Sampling:
  - train accuracy: 0.7731376975169301
  - test accuracy: 0.7363542739443872
  - Matriz de confusión:

1356	492
20	74

  - Área bajo la curva: 0.8154387722206872

6.9.3. MLPClassifier

Aplicamos K-Means al set de datos y luego MLPClassifier y los resultados obtenidos fueron:

- Aplico SMOTE:
  - train accuracy:0.7673941878692874
  - test accuracy: 0.7595262615859938
  - Matriz de confusión:

1404	444
23	71

  - Área bajo la curva: 0.8191604494795984
- Random Under Sampling:
  - train accuracy: 0.7516930022573364
  - test accuracy: 0.7605561277033985

- Matriz de confusión:

1405	443
22	72

- Área bajo la curva: 0.8195116054158607

6.10. Stacking

Lo que intentamos hacer en esta sección fue un ensamble de dos pasos, en el que en el primero usabamos los siguientes clasificadores:

- RandomForestClassifier
- ExtraTreesClassifier
- AdaBoostClassifier
- GradientBoostingClassifier
- XGBClassifier

Y como 2º paso MLPClassifier.

En el segundo paso predecimos con un promedio de las predicciones del primer paso. Esta ténica nos costó bastante y no nos dimos cuenta hasta muy tarde qué teníamos mal codificadas algunas cosas. Por otro lado, lo más difícil fue aplicar Grid Search a cada uno de los clasificadores para obtener los hiperparámetros óptimos.

La selección de features no fue sencilla, lo que hicimos fue un promedio de la importancia que nos devolvía cada clasificador del primer paso y luego seleccionar los 80 más importantes.

Los resultados obtenidos fueron:

Para esta técnica usamos Random Under Sampling:

- train accuracy: 0.7855530474040632
- test accuracy: 0.7554067971163748
- Matriz de confusión:

1394	454
21	73

- Área bajo la curva: 0.8524224002947407

7. Conclusiones

Para concluir, luego de haber probado muchísimos features y algoritmos, tanto para predecir, como para seleccionar features, balancear el set de datos, encontrar los mejores hiper-parámetros, etc; llegamos a que el algoritmo XGBoost es el que mejor predicción nos dio basándonos en el ROC AUC (métrica evaluada). Además utilizamos un algoritmo (RFE) que permite seleccionar los mejores K features del set de entramiento.