Universidad de Buenos Aires
Facultad de Ingeniería

2do Cuatrimestre - 2018

Organización de Datos (75.06)

Informe Trabajo Práctico 2

Predicting user conversions

Grupo 12: **Zulma LoDato**

Integrantes:

Nombre y Apellido	Padrón	Correo Electrónico
Donato, Juan Pablo	100839	juan.pablo.donato1996@gmail.com
Pistillo, Carolina Rocío	99177	caropistillo@gmail.com
Velasco, Ignacio	99796	iignaciovelasco@gmail.com

Fecha: 3 de diciembre de 2018

Link de **GitHub**: https://github.com/IgVelasco/TP2-Datos

Índice

1. Introducción general

El objetivo principal de este trabajo práctico fue el de probar distintos algoritmos de Machine Learning para predecir cuál es la probabilidad de conversión de un conjunto de usuarios seleccionados por Trocafone. Trocafone es una empresa de tecnología con operaciones en Brasil y Argentina. Implementan un modelo de negocio conocido como *ReCommerce* que plantea la compra, reacondicionamiento y venta de celulares previamente usados.

Para ello se cuenta con datos de eventos de *web analytics* de usuarios que visitaron la página www.trocafone.com, su plataforma e-commerce de Brasil. Los datos correspondientes a los eventos de *web analytics* fueron proporcionados en el archivo *"events_up_to_01062018.csv"*. Este set representa un subconjunto de la totalidad de eventos y dispone de una cantidad representativa de usuarios que visitaron la plataforma (hasta el 31/05/2018). Además, se cuenta con el archivo *"labels_training_set.csv"*, correspondiente a un subconjunto de los usuarios incluidos en el set de eventos, en el que se indica si los mismos realizaron o no una conversioón (label 1 o 0 respectivamente) desde el 01/06/2018 hasta el 15/06/2018. Este será el archivo utilizado para entrenar nuestros modelos de Machine Learning.

En primer lugar, se detallaran los features que pudieron obtenerse a partir de un preprocesamiento de los datos. Seguidamente se mencionarán los algoritmos utilizados y los modelos entrenados con los mismos, haciendo incapié en el desarrollo y los resultados obtenidos.

2. Preprocesamiento de los datos

Previo a utilizar los datos provistos, se procedió a realizar un preprocesamiento sobre los mismos para evitar utilizar los que sean poco representativos.

Habiendo realizado un análisis previo, pudimos observar que mas del 70 % de la información se encontraba en el ultimo mes. Además, solo aproximadamente 1600 personas (de todas las que se encontraban en los datos) no se metieron a la página en este mes y es importante tener en cuenta que son los datos mas actuales a la época que se nos pide realizar una predicción. Por ello, decidimos centrar nuestro análisis en su mayoría en el último mes de los datos. Sin embargo, no descartamos el resto de los datos para poder tener información sobre las personas que no interactuaron en el mes de Mayo.

Además, se dejaron de lado aquellas columnas en las que predominaban valores nulos o vacíos y aquellas que contenían datos irrelevantes para nuestro análisis, tales como *staticpage* o *screen resolution* que consideramos que, dada las características de dichas columnas, no aportaran información relevante en el análisis del trabajo.

A partir de estos datos, procedimos a extraer features que nos parecieron relevantes para realizar nuestras predicciones. Para ello tuvimos en cuenta, por ejemplo, la importancia para un usuario a la hora de realizar una conversión y todos los que nos permitieran obtener una mejor puntuación, es decir que pudieran ayudar a nuestros algoritmos a relacionar los usuarios y las conversiones entre sí de forma óptima.

Por otra parte, al hablar de features y sus características, nos pareció buena idea dividirlos según su tipo: *numéricos*, *categoricos* o *booleanos*. Llamaremos *features numéricos* a todos aquellos atributos que fueron extraidos por alguna relación matemática y cuyo resultado es, justamente, un número. Un ejemplo de este tipo podría ser la cantidad de eventos çheckout"que realizó una persona en el último mes, o el promedio de visitas que realiza una persona sobre un producto que le es de interés. Luego, tenemos algunos features *categóricos*, los cuales indican ciertas caracteristicas de los usuarios, como ser su país de procedencia, o el modelo en el que mayor eventos generó (podría ser el que mas le interesó). Por último, tenemos los features *booleanos*, que serán simplemente *1 ó 0* (True or False) dependiendo de la relación que se les imponga, por ejemplo, si una persona ingresó en el ultimo mes o no, si una persona visitó un producto mas veces que el promedio de visitas que tiene el producto por personas, etcétera.

A grandes rasgos, estos fueron los principales aspectos sobre nuestro preprocesamiento de datos. Lo que viene a continuación será un analisis con mayor detalle sobre los features utilizados (o no) para el entrenamiento

de los modelos de Machine Learning.

3. Feature Engineering

3.1. Features Armados

3.1.1. Features no usados

Los siguientes features, fueron probados en nuestro modelo y a lo largo del desarrollo del trabajo y no resultaron en una mejora en los resultados.

- Count vectors de modelos buscados y vistos teniendo en cuenta los mas buscados y mas vendidos.
- TF-IDF de la misma forma que el item anterior, observamos que tenia un peor desempeño que usando count vectors.
- Relaciones entre vistos sobre comprados:
  Utilizando count vector sobre la cantidad de veces que vio cada modelo y multiplicando a cada cantidad propia del usuario por la relación propia del modelo.
- Relación buscado sobre comprado:
  Igual que el anterior solo que en vez de vistos se utilizaron los buscados
- Numero de veces que realizo cada evento.
- Cantidad de meses distintos que ingresaron al sitio.
- Feature que indicaba si volvieron a meterse al sitio después de comprar.
- Cantidad de dias distintos que vio un mismo modelo.
- One hot encode con condition, color y storage.
  Creemos que es buena idea resaltar esta parte curiosa de nuestro trabajo. Este feature, al subirse a la competencia, nos bajo el puntaje considerablemente (aproximadamente 0,51), lo que nos hizo ver que estaba haciendo que nuestro modelo tuviera alto bias (estaba "*overfitteando*").

```
cv_val.tail(1)
```

	train-auc-mean	train-auc-std	test-auc-mean	test-auc-std
49	0.957888	0.001373	0.855803	0.004375

Figura 1: Resultado de la ultima iteración del cross validation. Score interno muy alto pero muy mal predictor

- One hot encode con dia de la semana y numero de la semana
- One hot encode con eventos por sesion
- *modelo_mas_visitado*: Modelo más visitado por un usuario en los 5 meses. Lo utilizamos en un principio pero luego notamos que dada la variedad de modelos posibles era conveniente utilizar la marca del modelo (feature categórico *marca_mas_buscada*).

3.1.2. Features usados

Como mencionamos en la sección anterior, decidimos agrupar los features según su tipo. A continuación se mencionan los mismos y en la sección 4 se explican las transformaciones realizadas a los datos para su obtención.

Features numéricos

- *Cantidad_visitas*: Cantidad de eventos *viewed product* por usuario para el modelo que más vio.
- *Promedio_visitas_producto*: Promedio de la cantidad de veces que un usuario visita un modelo distinto.
- *horas_mirando_productos*: Horas totales de un usuario realizando eventos de tipo *viewed product*
- *visitas_mes* **n** (n=1,2,3,4,5): Indica la cantidad de eventos que realizo un usuario en el mes n.
- *promedio_ingreso_mensual*: Promedio de la cantidad de eventos por mes para un usuario.
- *registros_semana***n:** Cantidad de eventos por persona para la semana n de los meses.
- *promedios_registros_diarias:* Promedio de la cantidad de visitas diarias.
- *dias_ingresando:* Cantidad de días en los que se registró algun evento para cada persona.
- *promedio_visitas_hora:* Promedio de la cantidad de visitas por hora
- *checkouts_ultima_semana:* Cantidad de eventos *checkout* realizados en la última semana. Este feature lo consideramos de importancia ya que si hizo muchos checkouts en la última semana probablemente no lo haga en las semanas siguientes.
- *cantidad_interacciones_por_modelo:* Indica por persona la cantidad de eventos para cada modelo para el mes 5.
- *cantidad_checkouts_por_modelo:* Indica por persona solo la cantidad de los eventos checkout para cada modelo para el mes 5.
- *Checkout_max:* Máxima cantidad de *cantidad_checkouts_por_modelo*
- *Checkout_mean:* Promedio de la *cantidad_checkouts_por_modelo*
- *cant_modelos_que_consulto_stock:* Cantidad de modelos distintos para los que hizo un evento *lead* en el mes 5.
- *diferencia_5:* Diferencia entre primer y ultima fecha del mes 5
- Por último, y para complementar, features sacados sobre la fecha utilizando la libreria "feature tools"

Otros features temporales:

Temporal de evento *checkout*

- Día de primer y ultima aparición del evento
- Diferencia entre primer y ultima aparición
- Sumatoria de checkouts en el mes
- Promedio de checkouts realizados por dia
  El promedio fue hecho con los días que realizo *checkout*

- Promedio de checkouts realizados por semana
  El promedio fue hecho con los semana que realizo *checkout*

Temporal de evento *conversion*

- Si realizo una compra en los ultimos 15 dias del mes

Temporal de evento *viewed*

- Sumatoria de viewed en el mes
- Promedio de viewed realizados por dia
  El promedio fue hecho con los días que realizo *viewed*
- Distribución standard de viewed realizados por dia

Features booleanos

- *busco_top_5_visitas*: Indica si el modelo más visto por la persona se corresponde a uno de los 5 modelos con más visitas de la región a la que pertenece la persona.
- *visito_mas_que_el_promedio*: Nos dice si la cantidad de visitas del usuario al modelo que mas vio es mayor a la cantidad de visitas promedio a ese modelo.
- *siempre_incrementando*: Para ver si una persona aumentó la cantidad de registros que realizó mes a mes.
- *inc_ultimo_mes*: Indica si hubo un incremento en la cantidad de eventos por persona en el ultimo mes respecto al mes anterior.
- *mismo_interes_ultimos_dos_meses*: Informa si el modelo más visto por un usuario en el último mes es igual al modelo más visto por un usuario en el mes anterior.
- *ingreso_el_ultimo_mes*: Indica si el usuario efectuó algún evento en el último mes.
- *ingreso_ultima_quincena*: Nos dice si el usuario efectuó algún evento en la última quincena.
- *mayor_actividad_ultima_semana:* Para ver si tuvo mayor actividad en la última semana respecto a la actividad en la primer semana.

Features categóricos

- *region_persona*: Región a la que "pertenece" cada persona, es decir, la región en que la persona realizó más eventos.
- *pais*: Pais correspondiente a la region a la que pertenece la persona.
- *evento_predominante_mes_5*: Tipo de evento que más generó en el último mes.
- *semana_mas_interactuante*: Semana del mes que más registros tiene de cada persona.
- *device_type*: Tipo de dispositivo desde el que más ingresó.
- *channel_frecuente*: Tipo de channel de donde proviene la mayor cantidad de eventos para cada persona.
- *marca_mas_buscada*: Marca del modelo más buscado por cada persona.

4. Transformaciones Realizadas

- *region_persona*: Este dato se obtuvo agrupando por persona, region y país y convervando la combinación que tuvo mayor cantidad de eventos para la misma persona. Esto último es debido a que algunas personas tuvieron ingreso desde mas de un lugar, pero conservamos aquel en donde la cantidad de registros es mayor.

- *Cantidad_visitas*: Obtenemos el modelo más visto por cada persona. Para ello filtramos por eventos *viewed product* y agrupamos por persona y modelo procediendo de la misma forma que antes para quedarnos con el modelo más visto por cada persona y la cantidad de visitas que va a ser nuestro feature.

- *region_persona, modelo_mas_visitado*: Hacemos un join de lo obtenido en *Cantidad_visitas* y en *most_visited_region* y obtenemos por persona: el modelo más visto y la región a la que pertenece cada persona.

- *busco_top_5_visitas*: Habiendo hecho esto, volvemos a los datos originales y nos quedamos solo con los datos de eventos *viewed product* y con las columnas correspondientes a la persona, región y modelo. Realizamos un join con el set de datos obtenido al principio para hallar el *most_visited_region*. Con esto, calculamos la cantidad de visitas por modelo y por región para quedarnos con un diccionario que tenga como clave la región y como valor una lista de los 5 productos con más eventos *viewed_product* de esa región.

  Por último, tomamos este diccionario y el DataFrame con el modelo más visto por cada persona y construimos nuestro feature *busco_top_5_visitas* que nos dice si el producto más visto por la persona es uno que entra en el top5 de los más vistos de su región.

- *Promedio_visitas_product*: Teniendo en cuenta el dataset generado para obtener *Cantidad_visitas*, obtenemos el promedio de visitas de un modelo por región agrupando por modelo y *most_visited_region*, sacando el promedio.

- *visito_mas_que_el_promedio*: Nos fijamos si dato del feature *Cantidad_visitas* es mayor que el del feature *Promedio_visitas_product*.

- *horas_mirando_productos:* Para ello nos quedamos con las columnas person y timestamp y filtamos para el evento *viewed_product*. Obtenemos el dia y mes de los eventos. Obtenemos un delta de tiempo con el que hayamos las horas por dia. Luego agrupamos por persona y sumamos las horas por dia.

- *visitas_mes***n:** Tomamos las columnas person, time y event. Hayamos el mes para cada fecha y agrupamos por persona y por mes contabilizando la cantidad de eventos de esa combinación. Luego creamos columnas para todos los meses del set de datos, donde cada una representará la cantidad de eventos que realizó una persona en ese mes.

- *siempre_incrementando:* Este feature será un booleano que indicará si siempre la cantidad de eventos por persona del mes siguiente es mayor a la del mes anterior.

- *inc_ultimo_mes:* Simplemente reutilizamos lo hecho para el feature anterior y obtenemos para cada persona si su cantidad de eventos en el mes 5 es mayor a su cantidad de eventos del mes 4.

- *promedio_ingreso_mensual:* Volvemos a reutilizar lo obtenido antes y realizamos un promedio de los eventos por usuario por mes, sumando las cantidades de todos los meses y dividiendolo por la cantidad de meses.

- *mismo_interes_ultimos_dos_meses:* Procedemos de la misma forma que al principio para hallar el modelo más visto por cada persona, pero esta vez para los ultimos dos meses. Luego comparamos si el modelo más visto del último mes es igual al del mes anterior.

- *evento_predominante_mes_5:* Para ello agrupamos por mes y nos quedamos solo con los del último mes. Luego agrupamos por pesona y evento y contabilizamos, quedandonos con el mayor numero de las distintas combinaciones.

- *semana_mas_interactuante, registros_semana***n:** Calculamos el número de semana segun el día del evento y agrupamos por número de semana y persona, sumando para obtener la cantidad de eventos para cada combinación. Luego creamos DataFrames para los registros de cada semana, hacemos un join de todo y nos quedamos con el nombre de la semana con mayor cantidad de eventos para cada persona.

- *device_type*: Me quedo con las columnas person, device_type y event del original. Agrupo por persona y dispositivo y cuento la cantidad de eventos relizados para cada combinación y me quedo con el mayor

- *channel_frecuente*: Me quedo con las columnas person, channel y event y procedo de la misma forma que para el feature anterior.

- *ingreso_el_ultimo_mes*: Ordenamos por mes de forma descendente y nos quedamos con el primero (ultimo mes de ingreso) para cada persona. Luego evaluamos si ese mes es igual a 5 (ultimo mes del DataFrame)

- *ingreso_ultima_quincena*: Procedo de la misma forma que antes, pero esta vez viendo si el numero de semana que ingreso es mayor a 21 (última quincena de todos los datos).

  En el repositorio pueden verse las subsiguientes transformaciones realizadas para la adición de features numéricos relacionados con factores temporales.

- *pais*: Este feature podria ser muy peligroso, porque el dataframe tiene demasiados paises, lo que implicaria muchas columnas de dummies a la hora de entrenar. Para reducirlas, mantuvimos paises importantes identificados en el primer trabajo práctico, como *Brazil, Argentina, Estados Unidos y Canadá*, y clasificamos al resto como *Another* ya que la relación de gente que proviene de los primeros países con respecto a los que no lo hacen es abismal.

- *marca_mas_buscada*: Al igual que el item anterior, conservar cada uno de los modelos distintos que Trocafone tiene en sus registros podría ser catastrófico. Para reducirlos, identificamos las marcas lideres en dichos dispositivos, como *iPhone, Samsung, Motorola y LG*, y clasificamos al resto como *Other*.

5. Algoritmos probados

A lo largo del trabajo se probó una gran cantidad de algoritmos, de los cuales algunos se termiaron descartando porque no mejoraban puntajes previos o no realizaban un aporte positivo al problema en cuestión.

Arboles de decisión

- XGBoost

  Considerado como uno de los mejores algoritmos de clasificación, el XGBoost implementa árboles de decisión diseñados para la velocidad y rendimiento. Es un algoritmo de Boosting y produce un árbol por iteración.

  Este algoritmo fue uno de los más utilizados y con los que individualmente mejores resultados se obtuvo antes de hacer un ensamble con otros. Fue uno de los primeros con los que empezamos a entrenar y al darnos tan buenos resultados lo mantuvimos desde un principio.

- Random Forest

  Combina o ensambla árboles de decisión en el momento de entrenamiento y devuelve el promedio (regresión) o la moda (clasificación) de los árboles individuales.

Lo utilizamos para ver la importancia de los features de las búsquedas y de los modelos pero lo terminamos descartando porque no obtuvimos mejores resultados.

- ExtraTrees Se utilizan con el objetivo de randomizar aún más la construcción de árboles y reducir la varianza producto de una mayor decorrelación entre árboles.

  Si bien es un algoritmo 'rápido', no generaba un gran aporte al resultado final, por lo que fue descartado.

- LightGBM

  A diferencia de otros algoritmos de gradient boosting, LightGBM construye el árbol hoja por hoja en lugar de nivel por nivel. Se debe tener en cuenta el parámetro de profundidad máxima para que no resulte en un árbol desbalanceado, ya que unos de sus usos principales es la velocidad.

  Es un algoritmo que, a comparación con el resto, es extremadamente veloz, y otorga muy buenos resultados.

- Gradient Boosting

  Esta técnica de Machine Learning para problemas de regresión y clasificación produce un modelo de predicción de forma escalonada como producto del ensamble de modelos de predicciones débiles, en general árboles de decisión.

  Su utilización en el trabajo nos sirvió mucho para complementar el trabajo de los otros algoritmos.

- MLP Classifier Algortimo provisto por SKLearn en su libreria *neuronal_network*. Es un Perceptrón multicapa que entrena iterativamente ya que en cada paso computa las derivadas parciales de su *loss function* con respecto a los parametros del modelo, para actualizarlos. Además, puede regularizar los datos para evitar overfitting.

Nearest neighboors

- KNN

  Este es uno de los algoritmos más simples y consiste en encontrar los k vecinos más cercanos para cada uno de los puntos que queremos clasificar. A partir de ello se puede predecir la probabilidad para la clasificación de una clase, teniendo en cuenta la cantidad de vecinos que pertenecen a la clase que se desea clasificar y normalizando por el total de vecinos (k).

  Al principio, lo utilizamos mucho en el trabajo ya que era sencillo, y obtenia buenos resultados. Mas adelante, con la aparición de los otros modelos, KNN fue descartado por no seguir al resto en cuanto a perforance y resultado.

Modelos lineales

- Linear Regression

  El modelo de regresión lineal resulta un modelo muy atractivo porque su representación es muy sencilla. Combina un conjunto específico de valores de entrada (x) cuya solución es la salida predicha para ese conjunto de valores de entrada (y). Tanto los valores de entrada como el valor de salida son numéricos.

- Linear Discriminant Analysis

  Este método permite encontrar una combinación lineal de features que caracteriza o separa dos o más tipos de eventos. La combinación resultante se puede usar como un clasificador lineal o, con mayor frecuencia, para la reducir dimensiones antes una clasificación.

- Perceptron

  Este algoritmo se basa en clasificadores binarios, es decir, una función que puede decir si una entrada pertenece o no a una clase específica. Es decir, genera un criterio para seleccionar un subgrupo a partir de un grupo de componentes más grande. Es un tipo de clasificador lineal porque realiza sus predicciones basadas en una función de predicción lineal que combina un conjunto de pesos con el vector del feature.

6. Algoritmo Final

En este apartado, explicaremos con detalle como decidimos nuestro algoritmo final.
Para comenzar, luego de combinar todos los features encontrados en un mismo *dataframe* para poder entrenar nuestro algoritmo, nos dimos cuenta que teníamos, en total, mas de 300 features. Esto era el resultado de combinar todos los features numéricos, booleanos y el "*one hot encoding*"de los features categóricos. Si bien el algoritmo final (que será detallado en breve),al entrenarlo con todos estos features, nos entregaba un resultado bueno (tanto interno como en Kaggle), tanta cantidad de features podía atraer un problema de dimensionalidad. Para solucionarlo, utilizando de la libreria *feature_selection* provista por SKLearn el algoritmo **RFECV**, el cual hace un ranking de features haciendo uso de eliminación de features recursiva y selección por *cross_validation*, obtuvimos el siguiente resultado:

```
from sklearn.feature_selection import RFECV

# The "accuracy" scoring is proportional to the number of correct classifications
clf_rf_4 = RandomForestClassifier()
rfecv = RFECV(estimator=clf_rf_4, step=1, cv=5,scoring='accuracy')   #5-fold cross-validation
%time rfecv = rfecv.fit(X_train, y_train)

print('Optimal number of features :', rfecv.n_features_)
print('Best features :', X_train.columns[rfecv.support_])
```

```
Wall time: 8min 55s
Optimal number of features : 223
Best features : Index(['busco_top_5_visitas', 'visito_mas_que_el_promedio', 'inc_ultimo_mes',
       'mismo_interes_ultimos_dos_meses', 'conversion_sum', 'conversion_mean',
       'viewed_sum', 'viewed_mean', 'viewed_std', 'dif_5_check',
       ...
       'PERCENTILE(checkout - viewed product)',
       'PERCENTILE(conversion - visited site)',
       'PERCENTILE(visited site - viewed product)',
       'PERCENTILE(conversion - checkout)',
       'PERCENTILE(conversion + visited site)',
       'PERCENTILE(checkout + conversion)',
       'PERCENTILE(conversion + viewed product)',
       'PERCENTILE(checkout + visited site)',
       'PERCENTILE(viewed product + visited site)',
       'PERCENTILE(checkout + viewed product)'],
      dtype='object', length=223)
```

Figura 2: Features más importantes por KFECV

Ahora, tendremos menos features, y comparando los resultados finales, existe una minima diferencia entre puntajes finales, y una amplia diferencia en cuanto al performance final.

Una vez seleccionados los features a utilizar, los combinamos por persona junto con sus *labels* y comenzaremos el entrenamiento. Dividimos nuestro sets de datos en set de entrenamiento y sets de test, y entrenamos los 4 modelos que decidimos finalmente a utilizar, en base a resultados obtenidos y performance de los mismos. Estos son:

- XGBoost
- Light Gradient Boosting Machine (LGBM)
- MLPClassifier
- GradientBoosting Classifier

Los modelos anteriores fueron elegidos desde todos los algoritmos mencionados en la sección anterior luego de repetidos intentos de entrenamiento y posterior obtención de resultados, que fueron subidos a Kaggle para testear que tan buena predicción se obtenia.
Una vez finalizado el entrenamiento de todos, debiamos combinarlos o **ensamblarlos**. Para ello, hicimos uso de la libreria **VotingClassifier** de SKLearn que nos permitirá ensamblar los modelos, otorgándole pesos a cada uno de ellos en base a los resultados obtenidos de cada uno.

Finalmente, utilizando el ensamble obtenido, y las personas a predecir junto con sus features, se obtiene el resultado final que es el que subimos en Kaggle.

7. Conclusiones

El trabajo práctico se basó, en gran parte, de nuestras capacidades para obtener features, de mayor o menor importancia, en base a los datos provistos por la competencia. Fue un trabajo interesante por la gran variedad y combinaciones que se podían extraer a partir de dichos datos, que si bien eran muchísimos, gran parte no eran de utilidad. Luego, es una tarea mas complicada aún la de decidir cuales de ellos son los mas importantes. En base a reiteradas experiencias de entrenar, obtener resultados, y con la ayuda de múltiples librerías provistas por SKLearn y otros, pudimos aprender como seleccionarlos. Por otro lado, este trabajo fue muy bueno y muy útil como para familiarizarnos con diferentes algoritmos de clasificación: probar múltiples librerías, entrenar diferentes algoritmos y rescatar diferencias de performance y resultados entre ellos. Podemos decir que entrenamos distintos modelos y en base a sus resultados, combinar algunos de ellos para un ensamble óptimo final. En este apartado también queremos destacar algunas cosas que nos hubiese gustado realizar pero no hemos podido, ya sea por falta de tiempo para estudiarlas en profundidad, o porque lo hayamos intentado erróneamente:

- Probar aún mas algoritmos de clasificación.
- Obtener features de sesiones de los usuarios. Llamamos sesiones a todas las filas del dataframe inicial de datos.
- Obtener features en base a 'ventanas de tiempo'.
- Separar las regiones de cada usuario en dos tipos, como ser regiones con alta cantidad de conversiones y regiones en las que no, etcétera.
- Vincular más y hacer más uso de la información que obtuvimos como resultado en el trabajo práctico 1. Si bien algo de esto fue considerado para este trabajo, por ejemplo, para determinar la importancia del pais proveniente del usuario o si el modelo que mas visitó o interactuó corresponde con el de mayor interés en su región, creemos que podríamos haberle dado un mayor uso.