Trabajo Práctico 2

Organización de Datos (75.06)

Universidad de Buenos Aires

Facultad de Ingeniería

Segundo Cuatrimestre - 2018

Grupo 39

Alumnos:

Prieto, Pablo (91561)

Zuretti, Agustín Santiago (95605)

Índice

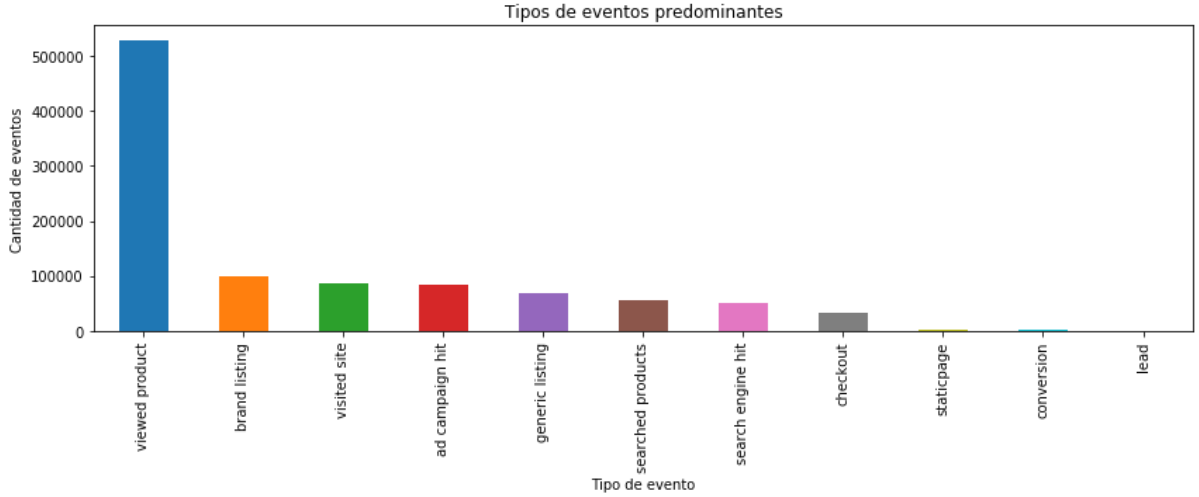
Introducción

El segundo trabajo práctico consiste en participar en una competencia de Machine Learning con el objetivo de estimar cuál es la probabilidad de que ese usuario realice una conversión en Trocafone en un periodo determinado.

Para comenzar la tarea disponemos del análisis exploratorio del set de datos que contiene información de todos los usuarios, realizado durante el primer trabajo práctico. Esto nos ayudó a conocer la información utilizada para intentar así predecir los resultados de la mejor manera posible.

También como punto de partida para realizar nuevos análisis orientados a el enunciado de este trabajo práctico.

Análisis del set de datos

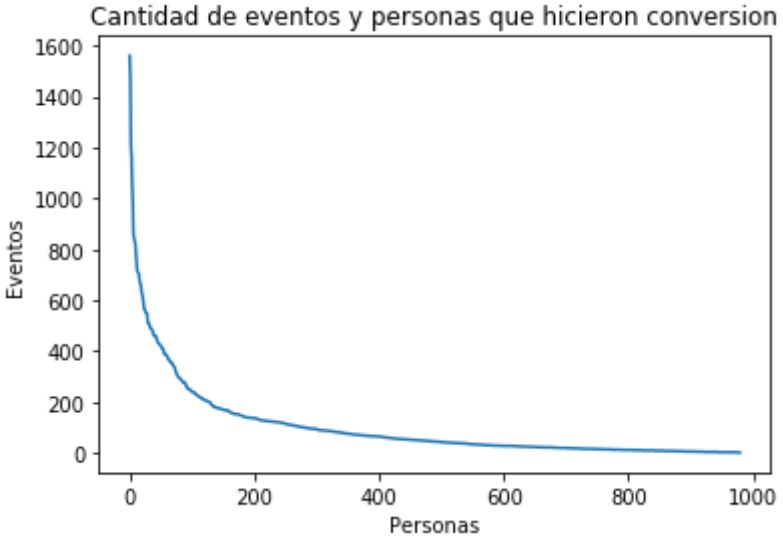


Como este trabajo se trata de predecir la probabilidad de conversión de un usuario, nos pareció interesante el análisis del set de datos realizado en el primer trabajo práctico.

En cuanto al set de datos del training, observamos que recibimos 19414 personas con sus correspondientes labels, de las cuales, solo 980 realizaban la conversión. Esta información

representa un porcentaje importante de las 1172 personas que realizaron una conversión en el set de informacion.

Esto nos aportó información muy importante ya que el set de datos se encontraba desbalanceado en la proporción de clases (muchos 0s y pocos 1s).



En los gráficos mostrados se observa la cantidad de eventos o filas que se generan luego de mergear el label training set con el events_up_to.

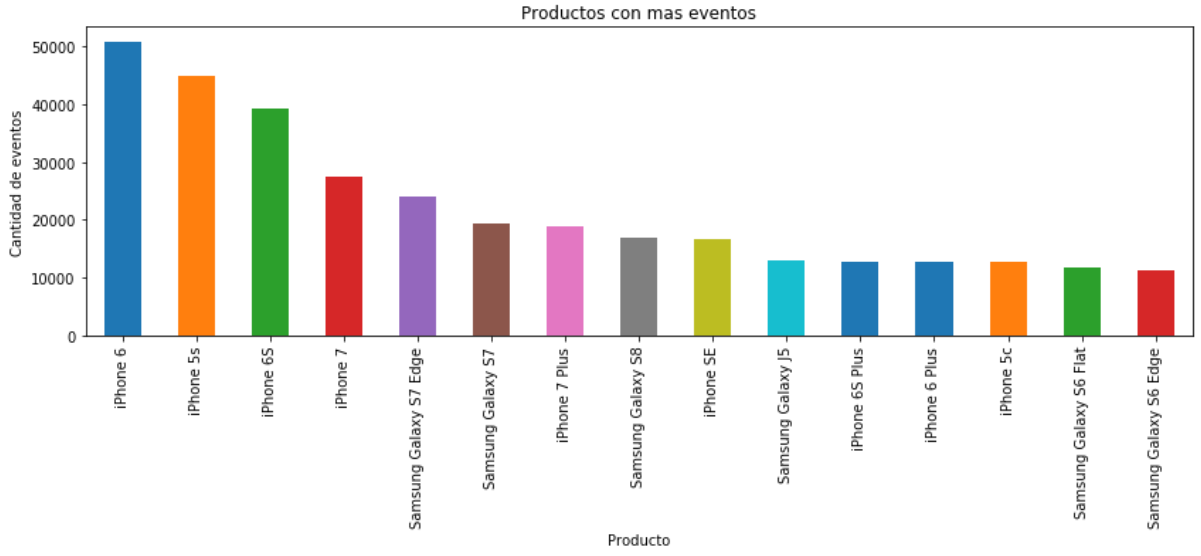
Por otra parte, según el análisis del TP1 pudimos identificar dos “perfiles” distintos entre productos y usuarios que están determinados por sus correspondientes eventos.

Los perfiles de productos están determinados por los eventos ‘conversión’, ‘checkout’ y ‘viewed product’. Las columnas relacionadas son: 'fecha', 'sku', 'model', 'condition', 'storage', 'color' (columnas analizadas en el TP1). Dichas columnas son las que forman parte del “perfil” del producto.

Así también, los perfiles de usuarios está determinado por el evento visited site. Las columnas relacionadas son: 'fecha' ,’person’, 'new_vs_returning', 'region' ,'country' ,'device_type' ,'screen_resolution' ,'operating_system_version', 'browser_version' (columnas analizadas en el TP1). Dichas columnas son las que forman parte del “perfil” del usuario.

Con estos perfiles se generaron los sets de datos de entrenamiento y predicción y se tuvieron en cuenta para la creación de features.

Además, observamos la cantidad de eventos relacionada a diferentes marcas y modelos y decidimos posteriormente crear un feature para cada una, ya que el tema principal del trabajo práctico trata sobre la conversión de estos.



Preprocesamiento

Filtrar el ruido de un set de datos, es una tarea muy importante ya que beneficia a varios algoritmos de machine Learning.
Por un lado observar cuántos eventos realizaban estas personas y por el otro aportar un peso balanceado que favorezca a la clase de 1 (conversión realizada). Si bien esto último tuvo mucho de “prueba y error” consideramos que en algunos algoritmos fue bastante beneficioso.
También consideramos como ruido a algunos features que no aportan ayuda alguna en nuestros scores.

En cuanto al feature engineering tomamos dos enfoques diferentes, pero en ambos casos, realizamos:

- Transformaciones del set tales como:
    - Label Encoder
    - One Hot Encoder
    - Normalización de los datos
    - quantile transformer
    - PCA

- Agregado de features basados en el análisis exploratorio y en los “perfiles”

En cuanto a quantile transformer, es un método que transforma las características para seguir una distribución uniforme o normal que tiende a extender los valores más frecuentes. También reduce el impacto de los valores atípicos (marginales): este es, por lo tanto, un esquema de preprocesamiento robusto. La transformación se aplica en cada función de forma independiente.
También aplicamos pca como técnica de reducción de dimensiones.

Para la generación del set de datos de entrenamiento y predicción se tuvieron en cuenta todos los eventos necesarios para encontrar los registros de todas las personas que se encuentran en los archivos de entrenamiento (labels_training_set) y de predicción (trocafone_kaggle_test). Luego del análisis, los eventos necesarios fueron: ‘conversión’, ‘checkout’ , ‘viewed product’, 'brand listing', 'generic listing', 'lead', 'staticpage' y ‘visited site'.

Por lo tanto, al armar los sets se tuvieron en cuenta los perfiles de productos para hacer un merge con los perfiles de usuarios y así tener los registros de las personas con todo el comportamiento realizado.

Para los productos, se separó entre productos comprados y vistos. Tal como vimos en el análisis exploratorio, los productos comprados son determinados por los eventos ‘conversión’ y ‘checkout’ y los vistos por el evento ‘viewed product’. Con estos, filtramos un dataframe con productos comprados, y otro con productos vistos.

Además, observamos que faltaban algunas personas que estaban solo en los eventos 'brand listing', 'generic listing', 'lead' y 'staticpage'. Con lo cual armamos un dataframe ‘otros’.

Para concluir, se armó el dataframe usuarios con el evento ‘visited site', y se realizó un merge con anteriores dataframe para obtener el set de datos de la forma:

	fecha_x	person	sku	model	condition	storage	color	compras_prod	vistas_prod	otros	fecha_y	new_vs_returning	region	country	device_
0	2018-05-18 00:11:56	4886f805	9288.0	Samsung Galaxy J7 Prime	Excelente	32GB	Dourado	1.0	0.0	0.0	2018-05-18 00:07:22	New	Rio de Janeiro	Brazil	Smartp
1	2018-05-22 20:29:35	0297fc1e	8992.0	iPhone 6S	Bom	32GB	Ouro Rosa	1.0	0.0	0.0	2018-05-02 01:28:27	Returning	Rio de Janeiro	Brazil	Smartp
2	2018-05-22 20:29:35	0297fc1e	8992.0	iPhone 6S	Bom	32GB	Ouro Rosa	1.0	0.0	0.0	2018-05-02 14:28:20	Returning	Rio de Janeiro	Brazil	Smartp
3	2018-05-22 20:29:35	0297fc1e	8992.0	iPhone 6S	Bom	32GB	Ouro Rosa	1.0	0.0	0.0	2018-05-03 01:55:52	Returning	Rio de Janeiro	Brazil	Smartp
4	2018-05-22 20:29:35	0297fc1e	8992.0	iPhone 6S	Bom	32GB	Ouro Rosa	1.0	0.0	0.0	2018-05-10 00:38:17	Returning	Rio de Janeiro	Brazil	Smartp

]:	›rod	vistas_prod	otros	fecha_y	new_vs_returning	region	country	device_type	screen_resolution	operating_system_version	browser_version	visitas_sitio
	1.0	0.0	0.0	2018-05-18 00:07:22	New	Rio de Janeiro	Brazil	Smartphone	360x640	Android 7	Chrome Mobile 64.0	1.0
	1.0	0.0	0.0	2018-05-02 01:28:27	Returning	Rio de Janeiro	Brazil	Smartphone	360x640	Android 6.0.1	Chrome Mobile 65.0	1.0
	1.0	0.0	0.0	2018-05-02 14:28:20	Returning	Rio de Janeiro	Brazil	Smartphone	360x640	Android 6.0.1	Chrome Mobile 65.0	1.0
	1.0	0.0	0.0	2018-05-03 01:55:52	Returning	Rio de Janeiro	Brazil	Smartphone	360x640	Android 6.0.1	Chrome Mobile 65.0	1.0
	1.0	0.0	0.0	2018-05-10 00:38:17	Returning	Rio de Janeiro	Brazil	Smartphone	360x640	Android 6.0.1	Chrome Mobile 65.0	1.0

Donde los features que se crearon fueron: ‘compras_prod’, ‘vistas_prod’, ‘otros’ y ‘visitas_sitios’. Nos basamos en la técnica de One Hot Encoder para partir los eventos en columnas, donde dichos features fueron asignados en 1 dependiendo en el dataframe donde se originó sino en 0 (por ejemplo si el dataframa es productos comprados, el feature ‘compras_prod’ es 1 y los features ‘vistas_prod’, ‘otros’ son 0, y para el feature ‘visitas_sitios’ el valor depende si hubo

coincidencias de personas en el merge con un left join entre los dataframes productos comprados y usuarios, entonces es 1 sino se rellena con 0).

Luego proseguimos a categorizar los features.
Usamos Label Encoder para transformar los valores de tipo string a enteros de todos los features excepto de 'fecha' (tanto de vistas o compras de productos como de visitas de usuarios), 'compras_prod', 'vistas_prod', 'otros' y 'visitas_sitios' (que ya estan categorizadas).
Sin embargo, para el feature 'model' se separó en marca y modelo, con lo que se obtuvo mejores resultados.

```
# Partimos el feature model en marca y modelo
sets['model'] = sets['model'].apply(lambda x: x.split(' '))
sets['marca'] = sets['model'].apply(lambda x: "apple" if str(x[0]) == "iPhone" else str(x[0]))
sets['modelo'] = sets['model'].apply(lambda x: ' '.join(x[1:]) if len(x)>1 else "0" )
sets.drop('model', axis=1, inplace=True)
sets.head()
```

```
#Categorizamos los valores de los features con label encoder
sets['sku'] = labelencoder.fit_transform(sets['sku'])
sets['marca'] = labelencoder.fit_transform(sets['marca'])
sets['modelo'] = labelencoder.fit_transform(sets['modelo'])
sets['condition'] = labelencoder.fit_transform(sets['condition'])
sets['storage'] = labelencoder.fit_transform(sets['storage'])
sets['color'] = labelencoder.fit_transform(sets['color'])
sets['new_vs_returning'] = labelencoder.fit_transform(sets['new_vs_returning'])
sets['region'] = labelencoder.fit_transform(sets['region'])
sets['country'] = labelencoder.fit_transform(sets['country'])
sets['device_type'] = labelencoder.fit_transform(sets['device_type'])
sets['screen_resolution'] = labelencoder.fit_transform(sets['screen_resolution'])
sets['operating_system_version'] = labelencoder.fit_transform(sets['operating_system_version'])
sets['browser_version'] = labelencoder.fit_transform(sets['browser_version'])
sets.head()
```

Además, el feature 'person' no se usó para entrenar sino solo para hacer merge, si se creó el feature 'person_int' que es el valor alfanumérico de 'person' transformado con Label Encoder a entero.
Pero tanto para el feature 'person_int' y 'sku' que son identificadores propios de persona y producto correspondientemente, no dieron buenos resultados, por lo que fueron descartados.

Para el feature 'fecha_x' (de producto) y 'fecha_y' de usuario que están en formato timestamp se categorizaron en un principio en mes, día, hora, minutos y segundos pero los últimos tres no dieron buenos resultados y se descartaron, y empezamos a usar dia de la semana.

```
: #Categorizamos la fecha del visto del producto en dia mes y dia de la semana  (ya que el anio es 2018)
  sets['dia_view_prod'] = sets['fecha_x'].apply(lambda x: x.day if type(x) != str else 0).astype('int')
  sets['mes_view_prod'] = sets['fecha_x'].apply(lambda x: x.month if type(x) != str else 0).astype('int')
  sets['dia_sem_view_prod'] = sets['fecha_x'].apply(lambda x: x.dayofweek if type(x) != str else 0).astype('int')
  sets = sets.drop('fecha_x',1)
```

```
: #Categorizamos la fecha de visita del usuario en dia mes y dia de la semana (ya que el anio es 2018)
  sets['dia_visit_user'] = sets['fecha_y'].apply(lambda x: x.day if type(x) != str else 0).astype('int')
  sets['mes_visit_user'] = sets['fecha_y'].apply(lambda x: x.month if type(x) != str else 0).astype('int')
  sets['dia_sem_visit_user'] = sets['fecha_y'].apply(lambda x: x.dayofweek if type(x) != str else 0).astype('int')
  sets = sets.drop('fecha_y',1)
```

Por lo tanto, el set generado es de la siguiente forma.

	person	sku	condition	storage	color	compras_prod	vistas_prod	otros	new_vs_returning	region	...	visitas_sitio	marca	modelo	person_int	c
0	4886f805	1622	3	4	28	1	0	0	1	78	...	1	6	67	5476	
1	0297fc1e	1572	1	4	34	1	0	0	2	78	...	1	8	13	189	
2	0297fc1e	1572	1	4	34	1	0	0	2	78	...	1	8	13	189	
3	0297fc1e	1572	1	4	34	1	0	0	2	78	...	1	8	13	189	
4	0297fc1e	1572	1	4	34	1	0	0	2	78	...	1	8	13	189	

g	region	...	visitas_sitio	marca	modelo	person_int	dia_view_prod	mes_view_prod	dia_sem_view_prod	dia_visit_user	mes_visit_user	dia_sem_visit_user
1	78	...	1	6	67	5476	18	5	4	18	5	4
2	78	...	1	8	13	189	22	5	1	2	5	2
2	78	...	1	8	13	189	22	5	1	2	5	2
2	78	...	1	8	13	189	22	5	1	3	5	3
2	78	...	1	8	13	189	22	5	1	10	5	3

Otro enfoque fue hacer un set de datos con pocas dimensiones, usando las features que consideramos más importantes, separando el campo model, en marca y producto, y considerando solo el dia de la fecha.
La idea de que este segundo set sea reducido era para ser implementarlo en un knn, el cual sufre mucho con la maldición de la dimensionalidad. También filtramos a las personas que tenían muchos eventos y no realizan una conversión ya que las consideramos como ruido, algo a tener en cuenta ya que knn es muy sensible a este.

Para la obtención de los hiperparametros utilizamos grid search, ambos métodos provistos por la librería Grid Search CV de scikit learn.

Algoritmos utilizados

Decision tree:

Decision tree consta de un Árbol con ramas en donde en cada una dividimos el set de datos de acuerdo a un cierto criterio con el objetivo de llegar a nodos hoja en los cuales podamos clasificar correctamente nuestros datos. Entre sus ventajas se destacan que es simple de entender, requieren poca preparación de los datos y tiene buena performance para datasets grandes. Entre sus desventajas se encuentra que un greedy para encontrar óptimos locales y además los Árboles demasiado complejos generan overfitting.
Una cosa que consideramos a medida que fuimos obteniendo resultados fue realizar cross validation con 4 k_folds. La misma nos ayudó a mejorar el overfitting.

Entre sus hiperparametros se destacan:
criterio: el criterio para determinar gini, o entropía, siendo este último la ganancia de *información*.
splitter : la estrategia para el split de cada nodo
max_depth: profundidad máxima de cada árbol
min_samples_split : el número mínimo de objetos para dividir un nodo interno

Random forest

Este algoritmo hace Bagging sobre árboles de decisión, en donde cada árbol usa un subconjunto de los atributos y hace un bootstrap del set de entrenamiento.

El Bagging es una técnica que disminuye la posibilidad de overfitting ya que cada clasificador no ve la totalidad de los registros del set de entrenamiento. Con esta idea probamos random
Sus hiperparametros son similares al anterior con el agregado de elegir la cantidad de árboles.

KNeighbors

KNN tiene 2 hiperparametros, la métrica a usar para calcular las distancias y el valor de k es decir cuantos vecinos vamos a considerar.

Es un algoritmo bastante interesante para probar, pero tuvimos en cuenta tanto que sufre del ruido en el set de datos como la maldición de la dimensionalidad. Esto último tiene que ver con que no usa índices espaciales y tiene que caer en una búsqueda por fuerza bruta cuando son muchas dimensiones.
Teniendo en cuenta esto probamos con ambos set de datos. Tanto la métrica a usar, como el valor de k, fueron obtenidos por grid search.

Multilabel perceptron

Para darle un enfoque diferente al modelo probamos también un perceptrón multicapa. La neurona o funcion de activacion elegida fue Sigmoid que en vez de 1 o 0 como valor de entrada (perceptrón) puede recibir cualquier valor en el medio. En cuanto al cálculo de sus neuronas utiliza.

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad con\ z = w\ .\ x + b$$

El funcionamiento de este algoritmo consiste en encontrar los pesos y bias para minimizar la función de costos. Stochastic Gradient Descent, no solo viene provista por la librería utilizada sino que estima el gradiente a obtener utilizando una pequeña muestra de inputs de entrenamiento seleccionados aleatoriamente y de esta forma reduce de gran manera el tiempo de la operación. Una mejora de esta llamada 'Adam' también fue probada.

También configuramos un Early Stopping, esto consiste en verificar la precisión de la clasificación con el set de validación al final de cada epoch, cuando vemos que esta precisión deja de mejorar entonces paramos y ahí tenemos la cantidad necesaria

Finalmente, en cuanto al número de hidden layers y la cantidad de neuronas por hidden layer, comenzaremos con 1 sola hidden layer con un número arbitrario de neuronas, por ejemplo 30, e iremos aumentando estas cantidades hasta encontrar un resultado que consideremos óptimo.

Obtención y Análisis de Resultados

En cuanto al análisis de resultados nos encontramos con varias dificultades desde el comienzo y tratamos de tomar decisiones coherentes apoyándonos en la teoría para la resolución de las mismas.
En los primeros pre procesamientos buscamos con grid search los hiperparametros y aplicamos knn y decision tree, los cuales nos arrojaban predicciones de entre 80 y 95 por ciento respectivamente en el set de validación, en estas instancias el mismo estaba separado en 75 para train y 25 para test. Al trasladar los resultados de kaggle vimos que en realidad estaba ocurriendo overfitting. Es decir, predice muy bien los datos con que pero no sirve para predecir nuevos datos.
Luego revisamos la cantidad de 0s y 1s tanto del set de validación como de la predicción y nos dimos cuenta de lo que estaba sucediendo.
En las predicción había una fuerte preponderancia de 0s por lo que lo que el “overfitting” no era eso realmente, sino que era producto de un desbalance en la proporción de clases del set de validación, por lo que si la predicción arrojaba solo 0s el score de validación iba a ser muy alto.
Ante este problema procedimos a entrenar el set con cross validation y así obtenemos resultados más parecidos a los que se trasladaron a kaggle.
En estos momentos incorporamos random forest, y una red neuronal para poder obtener mejores resultados. Al mismo tiempo incorporamos las features detalladas en la fase de preprocesamiento y procedimos a testearlas mediante prueba y error y grid search.
Por un lado el random forest, no presentó mejoras para ninguno de los 2 dataset, ni el validation set ni en kaggle.
Finalmente utilizamos un multilabel perceptrón, con una capa de 20 neuronas, y la funciona de activación de estas tanh, la cual funcionó mejor que ‘logistic’, que era la que habíamos planificado. el solver tampoco fue sgd, sino adam, una mejora de este.
El resultado fue parecido al de knn con K = 8 usando la métrica euclidiana. ambos métodos nos acercaron a un score de 70 en kaggle, siendo necesarios cerca de 60 submit en la plataforma.

Bibliografía

- Clase dictada por el profesor Pedro Domingos en la Universidad de Washington.
- Apunte de la materia
- Libro online sobre Redes Neuronales: http://neuralnetworksanddeeplearning.com/index.html
- Algoritmos usados https://scikit-learn.org/stable/supervised_learning.html
- link del repositorio:
- https://github.com/zurettiagustin/tp2-datos