FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico Nro. 2
Machine Learning

Eventos Trocafone

[7542] Organización de Datos
Segundo cuatrimestre de 2018

Alumno:	**Padrón**
CARRERO RIVEROS, Maria Daniela	99316
CANAVESE, Bautista	99714

Índice

1. Introducción

Mediante un set de eventos de usuarios dentro de la plataforma de Trocafone desde 01/01/2018 hasta 31/05/2018 provisto por la cátedra, se busca, mediante un proceso de Machine Learning, predecir la probabilidad de que el usuario vaya a realizar o no una conversión dentro de los próximos quince días.

2. Análisis Previo

Un análisis exploratorio previo nos dio la oportunidad de analizar el contexto en el que se está trabajando, los eventos más ocurrentes, qué productos se venden, de qué lugar provenían los eventos, entre otras cosas.

Lo que nos permitió directamente pasar al análisis de features, en donde surgieron las primeras preguntas: ¿Qué características de la persona nos iban a permitir predecir si compra o no? ¿Cuáles aportan suficiente información? ¿Qué tipo de algoritmos vamos a utilizar?

3. Ideas Iniciales

En principio, se propusieron features básicos tal como la cantidad de eventos por persona, pues lógicamente si alguien es más activo en la plataforma hay alta probabilidad de que realice una conversión. Otro ejemplo es la cantidad de veces que el usuario regresó al sitio, pues se puede interpretar como un comportamiento en el cual el usuario quiere volver porque algo le gustó que quisiera comprar, o puede ser un usuario que tiene compras frecuentes.

Por otro lado, otros features encontrados son basados en el tiempo. Se sacó importancia a la distancia en días al último evento de cada usuario, pensando en la posibilidad de que si el último evento de un usuario fue meses atrás, es probable que ese usuario no haga una conversión en los próximos días. También se calcula la cantidad de eventos en la última semana, pensando nuevamente en la posibilidad de un usuario con muchos eventos durante los últimos días es probable que esté próximo a realizar una conversión.

En cuanto a los algoritmos a utilizar, decidimos encarar por la parte de árboles de decisión, porque nos encontramos en un problema de clasificación, en donde debemos predecir si un usuario compró o no; también se puede pensar como una regresión entre cero y uno, pero consideramos que los árboles de decisión se adecuan más a nuestro problema.

Otro factor analizado fue la representación numérica de los datos, ya que teníamos por un lado muchas columnas categóricas (como el tipo de evento, región, país) y por otro la columna del tiempo de cada evento por persona, en donde se pueden sacar gran cantidad de features en relación con el mismo.

4. Desarrollo

El desarrollo del trabajo práctico consistió en varias etapas:

1. Etapa Temprana
2. Feature Engineering
3. Etapa Final

En cuanto a la etapa temprana se hicieron algunas pruebas con modelos simples llamados *baselines*. Estos no fueron muy buenos, pero nos permitieron comparar con los próximos modelos predictivos. Básicamente se realizaron predicciones con features básicos para familiarizarse con las herramientas y la plataforma Kaggle.

La etapa de feature engineering fue la que más tiempo consumió, consistió en crear buenos features en base a los eventos provistos. Esta etapa, en principio, fue bastante desordenada, probando varios modelos hasta llegar a un buen resultado y generar un buen set de entrenamiento, que es lo más importante para realizar una buena predicción.

Finalmente, en la etapa final, con el set de entrenamiento ya creado, se probaron distintos algoritmos como Random Forest, XGBoost, Light GBM, que fueron los que mejor resultado obtuvimos, también se testeó Adaboost + RF, entre otros que no resultaron óptimos para este problema.

4.1. Etapa Temprana

Durante esta etapa se realizaron las primeras pruebas con los datos y con la modalidad de Kaggle.

En un primer momento, consideramos crear un set de entrenamiento con los eventos por persona en promedio por mes o por semana, además de realizar un *hot-encoding* de cada evento por mes, y obtuvimos con Random Forest un primer score de 0.499. Visto que no es un buen score, decidimos cambiar el modelo a uno más simple, contando la cantidad de eventos distintos por persona y agregando features como el tiempo del último evento o la frecuencia de los últimos eventos del usuario, obteniendo así, para un Random Forest un score de 0.7.

Estos cambios comenzaban a ser significantes en nuestro modelo, agregando más features como las marcas más buscadas por el usuario, el país, la región y la ciudad de donde se realizó más eventos del usuario; con Random Forest se llegó a 0.83 y a partir de acá se empezaron a probar otros algoritmos y ensambles, como XGBoost que fue el algoritmo con mejor resultado.

4.2. Feature Engineering

El principal problema, como se mencionó anteriormente, fue la representación numérica de las tantas columnas categóricas y el manejo del tiempo.

En un principio, las variables categóricas se transformaron con *hot-encoding*, que además de dar malos resultados extiende innecesariamente las dimensiones del set de entrenamiento, por lo tanto se optó por un *mean-encoding* con suavizador e inclusión de ruido para disminuir el filtro de los labels hacia el modelo y no caer en Over-fitting.

En cuanto a la columna de tiempo probamos varias maneras de generar features, la que mejor nos funcionó fue tratar la distancia entre los eventos por días con respecto al último evento ocurrido en el set, así poder sacar relaciones entre los mismos en un determinado rango de tiempo bastante más sencillo.

Finalmente, durante el desarrollo del programa se fueron desarrollando distintos features que se dividen en grupos: información básica, información en base de tiempo, top eventos y el origen de los mismos.

4.2.1. Información Básica

Para considerar las preferencias de cada persona, se agregaron features como: la cantidad de eventos totales que generó en la plataforma y la cantidad por cada evento (conversión,

checkout, etc).

En la etapa temprana, sólo se tenía en cuenta los eventos totales de la persona, pero como vimos una mejora en el resultado, decidimos agregarlo también por categoría de cada uno.

También se agregaron features como: el tipo de dispositivo más utilizado por el usuario, y la cantidad de veces que éste regresó a la plataforma.

4.2.2. Información en Base de Tiempo

Probamos varias maneras de como extraer features en base al tiempo, en un principio utilizamos un Random Forest como método de de testeo para extraer los más relevantes; la distancia en días del último evento del usuario, distancia en días al primer evento y la frecuencia promedio en días de los eventos de las últimas dos semanas, fueron los elegidos finalmente para el modelo. Otros como la cantidad de eventos en promedio por mes o el mes en el cual se realizó cada evento, no fueron los mejores.

4.2.3. Top Eventos

En principio tuvimos en cuenta el evento más frecuente por persona, lo cual nos da una idea del tipo de usuario que es dentro de la plataforma, si es un usuario que sólo mira productos o compra directamente.

Luego como observamos buenos resultados, decidimos agregar la marca, el modelo y el estado más visitado por el usuario, esto caracteriza a la persona con qué tipo de producto es de su preferencia, si prefiere productos de gama baja/media/alta por ejemplo.

4.2.4. Origen de Eventos

El origen de los eventos está compuesto por: región, país y ciudad.

En cada caso hay más de 100 tipos de variables categorías, por lo que un *hot-encoding* no seria lo mejor, como se dijo anteriormente, se utilizó un *mean-concoding* para codificar estos features.

4.3. Etapa Final

Una vez desarrollado y pulido el set de entrenamiento, se probaron, principalmente algoritmos en base a árboles de decisión.

Dentro de los algoritmos basados en árboles, un conocido ensamble Random Forest. El mismo está basado en seleccionar features al azar, realizar una predicción con los mismos y finalmente con los resultados de todos los árboles realizar la predicción final. Con este algoritmo logramos un score máximo de 0.83, que no fue el óptimo para el problema.

Luego probamos XGBoost con buenos resultados, similares a los de Random Forest. Luego de realizar varias pruebas, pasamos a la etapa de optimización de hiper-parámetros, que fue bastante desordenada, además de probar mas features y otros algoritmos como LGBM, Adaboost, entre otros.

No veíamos un avance relevante en la predicción y tampoco una buena referencia con la que comparar los modelos, ya que la diferencia entre los mismos era insignificante. Por lo tanto se propuso utilizar el método de *Grid Search K-Fold Cross Validation*, consiste en tomar una cantidad finita de hiper-parámetros, dividir el set de entrenamiento en K partes iguales (tomamos K=5), tomar todas las combinaciones de cuatro y predecir con la restante, luego

tomar el promedio, en este caso, de las cinco predicciones; así para cualquier combinación de hiper-parámetros. Finalmente utilizando este método llegamos a un tope de score de 0.856 para un XGBoost.

En este punto, se descubrieron nuevos features observando la importancia de los mismos a la hora de la predicción, en nuestro caso la cantidad de eventos distintos por usuario dio buen resultado, por lo tanto decidimos agregar la distancia en días al último evento distinto para cada usuario; dando un score de 0.876 para el mismo algoritmo.

4.3.1. Entrenamiento Random Forest

Random Forest fue el primer algoritmo que se probo, siendo el que generalmente funciona bien en la mayoría de casos. En nuestro caso nos dio muy buenos resultados, pero finalmente no fue el que mejor funciono.

Se utilizo mas que nada para obtener y comparar la importancia de features, ya que dentro de todo es bastante estable en ese sentido.

El score final alcanzado con este algoritmo para 1000 estimadores es de: 0.8300.

4.3.2. Entrenamiento XGBoost

Primer algoritmo de gradient boosting que decidimos probar fue XGBoost, por su fama en competencias de machine learning, además de adaptarse a nuestro problema. En un principio sin algunos features totalmente pulidos, la diferencia entre este y LGBM era mínima, rondando por 0.8560 ambos.

Luego de los retoques mencionados anteriormente, la diferencia si hizo notoria, alcanzando nuestro mejor score: 0.8760.

4.3.3. Entrenamiento LGBM

Otro algoritmo de gradient boosting que decidimos probar fue Light GBM, supero la prediccion realizada con Random Forest, pero no a XGBoost, alcanzando un score final de 0.8709.

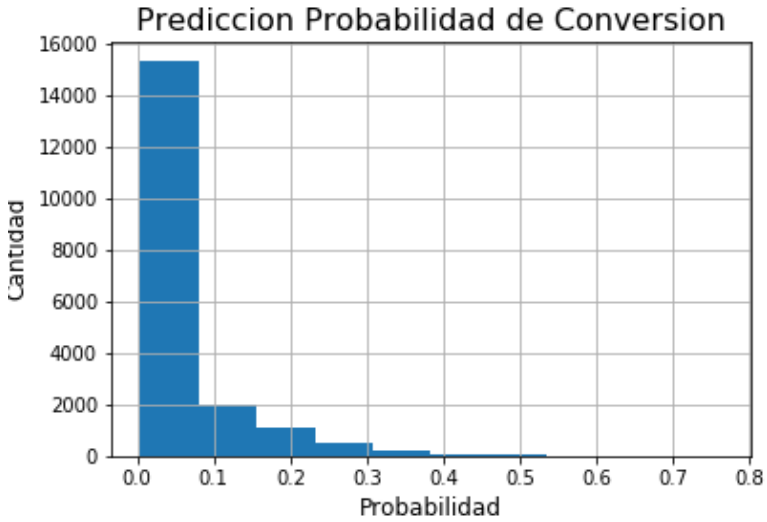
4.3.4. Otros Entrenamientos

Ademas de los algoritmos antes mencionados, se probaron distintos ensambles y metodos de boosting, como Adaboost + RF que no fueron optimos para el problema.

5. Submit Final

Elegimos XGBoost como submit final, a continuación se mostrara un breve resumen de los resultados en prueba.

Test	**roc_auc**
K-Fold Cross Validation	0.87202
Kaggle	0.87634



6. Problemas a resolver

Hemos tenido un único inconveniente, a la hora de aplicar label-econding sobre los features categóricos, como se menciono anteriormente, se agrega un poco de ruido; esto es indispensable para no caer en Over-Fitting. Por lo tanto al generar nuevamente el set de entrenamiento/test no podemos obtener exactamente la misma predicción.

Probamos realizarlo sin ruido pero, no fue lo mejor, obtuvimos una buena predicción para el set de prueba, pero mala para la predicción final (caso de overfitting). Además de testear otros metodos como hot-encoding, que tampoco funcionaron de la mejor manera, por lo que decidimos dejarlo como está.

7. Conclusión

La primera conclusión que extraemos es la gran importancia de definir un buen set de entrenamiento, y no por bueno debe ser complejo, en nuestro caso un modelo más simple nos llevó a mejores resultados, aumentando en un 0.3 la precisión sin hacer ninguna modificación sobre el algoritmo utilizado.

En segundo lugar, observamos que a priori no se puede saber qué algoritmo será el óptimo para el modelo, el hecho de probar distintos algoritmos combinando con distintos modelos hace un panorama más amplio al problema y se pueda buscar la solución que más convenga para el modelo.

Finalmente, decidimos elegir XGBoost como nuestro algoritmo de predicción final, ya que principalmente fue el que mejor resultado nos dio, además de funcionar bastante bien en cuanto a performance, más que nada ejecutando el programa desde una PC con recursos limitados.

8. Más información

Nuestros análisis fueron realizados en Python. Utilizamos un repositorio público de github para juntar análisis y filtrar los que creíamos eran adecuados para incluir en este informe. El link del repositorio es:

`https://github.com/bauticanavese/Datos-Tps`

Dentro del repositorio se encuentra una carpeta llamada TP2, en donde se encuentra todo el procedimiento descripto en el trabajo práctico.

Referencias

[1] Apunte Cátedra

[2] Kaggle, *discusiones y notebooks*

[3] Sklearn, *https://scikit-learn.org/stable/* Doc. de algoritmos.

[4] XGBoost, *https://xgboost.readthedocs.io/en/latest/* Doc. de XGBoost.