FACULTAD DE INGENIERIA
Universidad de Buenos Aires

FIUBA - 75.06

Organización de Datos

Trabajo práctico 2, Grupo 28 (Macri Dato)
1er cuatrimestre, 2019

Integrantes:

Nombre	Padrón
GAIDO, Nicolás	100856
MARCÓ DEL PONT, Matías	101302
ROMERO VÁZQUEZ, Maximiliano	99118

Fecha de Entrega: 24/06/2019

Link de GitHub:

https://github.com/MatiasMdelP/Organizacion-de-Datos-TP2

Índice

1. Objetivo

El objetivo del informe es poder explicar los criterios utilizados para la aplicación de los distintos algoritmos de Machine Learning para lograr estimar, para un instante dado[1]:

- $S_t(d)$**:** el tiempo hasta que un dispositivo (d) aparezca nuevamente en una subasta RTB (*Real-time bidding*[2]).
- $S_c(d)$**:** el tiempo hasta que un dispositivo (d) convierta.

2. Introducción

Este trabajo práctico consiste en aplicar distintos algoritmos de *Machine Learning* para poder determinar, para cada dispositvo presentado por la empresa ***Jampp***, el tiempo que transcurrirá hasta que el mismo aparezca nuevamente en una subasta, y el tiempo hasta que el usuario del mismo decida instalar una nueva aplicación. Para luego poder participar de una competencia con los demás grupos presentes en la materia.

El set de entrenamientos consiste en un total de 4 dataset distintos provistos por la empresa mencionada anteriormente. Allí se pueden encontrar datos acerca de clicks de publicidades mostradas (*clicks.csv*), de eventos intra-app (*events.csv*), de instalaciones de aplicaciones a partir de publicidades mostradas (*installs.csv*), y de subastas RTB (*auctions.csv*).

3. Búsqueda de Features

Para entrar en tema, empezaremos detallando los diferentes *features* obtenidos luego de realizar *feature engineering*. Es decir, aquellas características que consideramos que son relevantes con respecto a aquello que queremos estimar, y hacen que los algoritmos de machine learning puedan funcionar.

Para poder llevar a cabo dicha búsqueda, se tuvo en consideración los resultados obtenidos de la realización del análisis exploratorio de los datos durante el trabajo práctico número 1, y sus respectivas conclusiones. La única diferencia entre el conjunto de datos presente en éste trabajo con el anterior, es la fecha a la que pertenecen. En ésta oportunidad corresponde al período del 18 de Abril al 26 de Abril de 2019, mientras que para el anterior iba del 5 de Marzo al 13 de Marzo del mismo año.

El primer paso que se realizamos fue separar los datos según una ventana de tres días, ya que los datos objetivos corresponden al período entre el 28 y el 30 de Abril. Es decir, el máximo valor es de 3 días.

Por ende, tendremos un total de cinco ventanas para el entrenamiento, que se presentan a continuación[3]:

- Del 18 al 20.
- Del 19 al 21.
- Del 20 al 22.
- Del 21 al 23.
- Del 22 al 24.

Cada features fue calculado para cada dispositivo presente en cada uno de los datasets mencionados en la sección 2, y a su vez, por cada ventana. Los features finales usados fueron los siguientes:

[1] Ambos tiempos se miden desde las 0:00 del 27 de Abril, en segundos.
[2] Ofertas en tiempo real.
[3] Todos corresponden a días de Abril de 2019.

- ***Count*:** cantidad de veces que un mismo dispositivo realiza una determinada *acción*. Dicha acción toma un nombre según el dataset que estamos analizando.

  Por ejemplo, si nos encontramos en el archivo *installs.csv*, el feature en cuestión sería *installsCount* que representaría la cantidad de instalaciones que realizó el usuario. Lo mismo ocurrirá con las siguientes características.

- ***MostFreqDay*:** calcula la máxima cantidad de registros por día.

- ***MeanInterval*:** corresponde al promedio de intervalos entre dos acciones realizadas por el mismo usuario.

- ***AproxFreq*:** frecuencia aproximada entre dos acciones.

- ***StdDevInterval*:** intervalo de tiempo hasta que un mismo dispositivo aparezca nuevamente.

- ***Last*:** equivale al tiempo (en segundos) hasta la última acción registrada por el dispositivo.

Para finalizar con este etapa, todos los features de todas las ventanas y datasets fueron concatenados en un mismo archivo en formato csv.

Además de probar con estos features numéricos, se buscó si al utilizar features del tipo categórico podríamos obtener mejores resultados pero esta implementación no fue mejor sino que la empeoró. Por ende se decidió usar solamente los features mencionados en los items anteriores.

Algunos de las características categóricas que resultaron sin éxito fueron: si la acción se realizó mediante conexión wifi, según el tipo de conexión utilizada al momento de realizarla, entre otras.

4. Modelos utilizados

En ésta sección mostraremos todos los modelos/algoritmos utilizados durante todo el proceso de entrenamiento de nuestro algoritmos de Machine Learning. Algunos de estos modelos se usaron para poder llegar a mejores resultados en cuanto a las predicciones que tienen como objetivo el presente trabajo, mientras que otros, se utilizaron solamente para ver como se comportaba nuestro sistema.

4.1. Survival Analysis

Survival Analysis, o también conocido mediante su traducción como *análisis de supervivencia*, generalmente se define como un conjunto de métodos para analizar datos donde la variable de resultado es el tiempo hasta la ocurrencia de un evento de interés.

En análisis de supervivencia, si no se llega a registrar un evento[4] (porque esta fuera de nuestra ventana de 3 días) se dice que esta censurado, para ello se creó una columna especial denominada ***uncensored***[5]. Para cualquier análisis que se quiera hacer, se necesita saber que evento se registró ($uncensored = True$) y cual no ($uncensored = False$).

Luego para los eventos censurados, el valor que importa es la última vez que se supo de ese dispositivo. En este caso, se sabe que hasta la última hora del tercer dia el evento no se produjo.

El primer modelo elegido fue ***Cox proportional hazard's model*** ya que nos pareció que era el más simple como para comenzar. Dicho modelo requiere que los labels utilizados sean un *'structured array'* de python. Por suerte esto no fue un inconveniente ya que existe una libreria provee funciones para obtener estos a partir de un DataFrame de Pandas como teneníamos inicialmente. Por otra parte, nos devuelve un coeficiente para cada feature llamado *Log Hazard Ratio*.

Es importante aclarar que el modelo no predice el valor S_t que queremos, sino un valor de *riesgo* del dispositivo, es decir, para cada instante t, cual es el riesgo de que suceda el evento.

[4] En estos casos, llamamos *evento* al hecho en interés/estudio y no a cualquier tipo de acción categorizada dentro de una aplicación.

[5] Sin censura.

La performance se mide con un índice que se fija que el orden corresponda con los labels. Esto es, que si a tiene mayor riesgo que b, entonces a debe haber sucedido primero.

Si llegamos a como resultado a un *0* indica que se trata de un modelo totalmente contrario a la realidad, *0,5* un modelo aleatorio y *1* un modelo perfecto.

Luego de haber realizado este modelo para la primera ventana del archivo que contiene las subastas llegamos como resultado del entrenamiento:

$$(0{,}7402916757320268, 31327425, 10990253)$$

Mientras que al realizarlo con el set de test:

$$(0{,}7307764723148017, 31151509, 11476449)$$

En donde el primer valor es el mencionado anteriormente, mientras que el segundo hace referencia a la cantidad de pares que concordaron, y el tercero a la cantidad de pares que no.

4.2. Ridge regression

Ridge regression (Regresión Ridge o regularización de Tikhonov), es un modelo que resuelve un modelo de regresión donde la función de pérdida es la función lineal de cuadrados mínimos, y la regularización viene dada por la norma euclídea (L2).

Por otra parte dicho modelos puede mejorar los errores de predicción al reducir en tamaño los coeficientes de regresión que sean demasiado grades para reducir el sobreajuste (*overfitting*), pero no realiza selección de variables y por tanto no produce un modelo más interpretable.

Subiendo este modelo a la plataforma de competición *Kaggle* obtuvimos como resultado un score de 141120.37182, siendo este, el mejor utilizando regresión Ridge.

4.3. Lasso

Lasso (Least Absolute Shrinkage and Selection Operator) es un método de análisis de regresión que realiza selección de variables y regularización para mejorar la exactitud e interpretabilidad del modelo estadístico.

Es un modelo muy similar conceptualmente a la regresión Ridge, agregando una penalización para los coeficientes distintos de cero. Pero a diferencia de Ridge que penaliza la suma de los coeficientes cuadrados, Lasso penaliza la suma de sus valores absolutos.

Realizamos varios predicciones con este modelo obteniendo como peor resultado un score de 825950.79 y el mejor de 128158.75806. El score representa al error cuadrático medio.

Otro resultado obtenido con Lasso fue 130568.40202.

4.4. Elastic Net

Elastic Net es un modelo que trata de combinar las penalizaciones de *Ridge regression* y *Lasso* para poder obtener lo mejor de ambos modelos. Utilizando este modelo se obtuvo un puntaje de 132374.79459.

Como se estudió en las clases teóricas de la materia, si realizamos un ensamble de los modelos podemos llagar a mejores resultados que al utilizarlos por separado. Realizar un ensamble de los distintos algoritmos quiere decir concatenarlos.

Luego de hacer un ensamble entre ***Ridge Regression***, ***Lasso*** *y* ***Elastic Net*** obtuvimos un score de 128752.30140. Como podemos apreciar, en este caso no logramos el resultado más óptimo, ya que como mencionamos en la sección 4.3, realizando las predicciones con el modelo *Lasso* solamente, llegamos a un mejor puntaje en Kaggle.

También fuimos modificando como tratamos a los valores nulos. Primordialmente los reemplazamos por cero, y luego por un promedio de toda la columna a la que pertence, obteniendo mejores resultados con esta última convención.

4.5. XGBoost

XGBoost es una implementación de código abierto popular y eficiente del algoritmo de árboles aumentados de gradientes. La potenciación de gradientes es un algoritmo de aprendizaje supervisado que intenta predecir de forma apropiada una variable de destino mediante la combinación de estimaciones de un conjunto de modelos más simples y más débiles.

Simplificando, podemos decir que XGBoost está netamente basado en el estado del arte en clasificación, y realiza un Boosting de árboles de decisión.

Al subir este modelo a la plataforma de competición obtuvimos puntajes en el rango entre 142702.52405 y 145636.03102. Como podemos ver, de todas maneras, quedamos detras de nuestro mejor puntaje hasta el momento.

Otra característica de éste método es que nos da un indicio de aquellos features que son mejores y cuales no funcionan tan bien como lo esperado. Por ello, como nuestro mejor puntaje ha sido mediante la utilización del modelo ***Lasso***, volvimos a correr éste último algoritmo, pero con nuestro mejores features, según el algoritmos de XGBoost. Con ello, obtuvimos, en *Kaggle*, un score de 129686.53168, que sigue siendo peor que lo calculado con todos los features.

5. Conclusiones

El informe trata de ilustrar todo el procedimiento realizado en la elaboración del trabajo práctico número 2 de la materia Organización de Datos, así como también los resultados o scores obtenidos.

El mejor puntaje que obtuvimos en la competencia en Kaggle fue con el modelo ***Lasso*** con un resultado de **128158.7580**.

Creemos que para poder obtener unos mejores resultados tendríamos que econtrar mejores features que los utilizados para entrenar nuestro sistema. Lamentablemente se usaron los 6 features numéricos mencionados en la sección 3, no porque estos son los mejores, si no que los demás que encontramos hacía que nuestros algoritmos funcionen todavía peor.

Otras soluciones al problema de mejorar el puntaje, puede ser la utilización de otros modelos predictivos y una mejor búsqueda de los hiperparámetros que requieren los algoritmos mencionados en este informe.