Universidad Austral - Facultad de Ingeniería
Maestría en Data Mining
Laboratorio de Implementacion I, Clases Gustavo Denicolay

Tarea para el Hogar 2021-10-01

Esta tarea para el hogar está dedicada a todos los alumnos que cursaron con el profesor Gustavo Denicolay la séptima clase de la materia, el viernes 01 de octubre de 2021

La idea de esta tarea para el hogar es

- incorporar el script `811_dataset_epic.r` que permite generar distintos datasets mediante el sencillo concepto de *palancas*, teniendo en cuenta el Data Drifting, Feature Engineering de variables del mismo mes e históricas y reducir el tamaño del dataset utilizando *canaritos*.
- generar corridas con `822_epic.r` que permiten rápidamente entrenar en distintos períodos y generan una salida para luego poder hacer stacking. La velocidad proviene de que no hacen cross validation y que para reducir el dataset hacen undersampling de la clase mayoritaria los "CONTINUA"
- generación de dataset para stacking con `834_dataset_stacking.r`
- modelos de stacking puro `845_stacking.r`

El algoritmo que estamos utilizando es el LightGBM
La clase que estamos utilizando es la binaria2 pos={BAJA+2, BAJA+1} , neg={CONTINUA}

Todas las corridas se harán en Google Cloud, al comienzo de cada script está una recomendación de que tamaño de máquina virtual utilizar.

Usted deberá ir completando la planilla `andamios.ods`

1. Prerrequisito Storage Bucket de Google Cloud

Conéctese en su navegador al Google Cloud Console https://console.cloud.google.com/
Vaya al Cloud Storage Bowser https://console.cloud.google.com/storage/browser

Allí deberá ver su bucket, haga click sobre su bucket y navegando deberá ver el siguiente contenido

- `datasets`
- `datasetsOri`
  - `paquete_premium.csv.gz`
- `exp`
- `kaggle`
- `log`
- `modelitos` #verificar que se tiene esta carpeta !!
- `work`

2. Prerrequisito Imagen de la máquina Virtual

Desde el browser que tiene conectado a Google Cloud vaya al link
https://console.cloud.google.com/compute/images

y verifique que tiene la imagen `image-dm`

3. Actualización en su PC local de su repositorio

Primero actualice SU repositorio github con el oficial de la materia, si no recuerda como hacerlo siga este instructivo https://docs.github.com/es/github/collaborating-with-pull-requests/working-with-forks/syncing-a-fork

Luego vaya a su PC local y actualice la copia que tiene en su pc local de su repositorio GitHub, con el comando `git pull`
En particular abra en su PC local la planilla Andamios`.ods` la irá completando con las siguientes corridas, se encuentra en la carpeta `labo2021 / clasesGustavo / TareasHogar / Tarea20210924`

4. Script `811_dataset_epic.r`

A partir de ahora refinaremos nuestra metodología. Tendremos un script con el que se genera el dataset y otro script que toma ese dataset, no lo modifica ni le crea variables nuevas, ni siquiera lags, y construye sobre él el mejor modelo posible con Bayesian Optimization y LightGBM.

En este script `811_dataset_epic.r` unicamente construimos el script. Lea en gran detalle este script, es fácil de seguir. Usted deberá agregarle código, todo lo que ha escrito de nuevas variables lo debe agregar a la funcion `AgregarVariables` en donde dice `#Aqui debe usted agregar sus propias nuevas variables`

Al inicio del script están las *palancas* , desde donde se controla lo que hace el script, las palancas son generalmente switches que usted puede poner en TRUE o en FALSE , otras veces son un vector de valores.
Poner en TRUE implica que enciende esa palanca y que se agregrán nuevas columnas al dataset, sea cuidadoso con la cantidad de palancas que enciende, ya que dataset deberá ser manejable.

Cada vez que usted corra este script para que se genere un dataset con un nombre distinto deberá cambiar la linea 25
`palancas$version <- "v002"`
a una nueva versión. Es muy importante tener este versionado rudimentario de los datasets, que no puede hacerse por git.

Es de gran importancia la palanca
`palancas$canaritosimportancia <- TRUE`
esta palanca agrega canaritos al dataset, corre un LightGBM con unos parámetros razonables, obtiene la importancia de variables, y luego elimina las variables que no son importantes (las que están por debajo de la capa geológica de canaritos). De esta forma el tamaño del dataset se reduce a la mitad. Piense que extraño es esto, en la generación del dataset corremos un rápido modelo predictivo con el solo propósito de reducir la cantidad de variables y de esa forma hacer que lo que siga corra más rápido. Este rápido modelo no tiene nada que ver con los que se construirán lluego.

Lea en gran detalle el script, es muy facil de seguir. Si usted no tiene conocimientos previos de programación, NO INTENTE entender en este momento que hace la función `fhistC` escrita en lenguaje C . tómela como una caja negra y avance. En los meses de Enero y Febrero, frente a la montaña o el mar, con un buen desayuno sobre la mesa podrá buscar la inspiración para hacerlo.

Finalmente se genera el dataset `/datasets/dataset_epic_v002.csv.gz`

5. Script 822_epic.r

Este script toma como input el dataset generado en el script anterior, el 811 . El dataset a utilizar se controla con la linea

```
karch_dataset     <- "./datasets/dataset_epic_v002.csv.gz"    #este dataset se genero en el script 811_dataset_epic.r
```

El script 812_epic.r es realmente complejo, intenta llevar a cabo una gran cantidad de acciones en forma cuidadosa y sutil.

El objetivo fundamental de este script ha sido acelerar los tiempos de corrida, recurriendo para ello a ciertos compromisos ingenieriles entre la calidad de la predicción y la velocidad de corrida. El objetivo final es que reduciendo los tiempos de corrida se podrán nuevos experimentos.

Los tres puntos más destacables de este script son:

- no hace 5-fold cross validation, sino que entrena en ciertos meses y testea en otros.
- no entrena sobre todos los registros, sino que arma un dataset a donde van todos los BAJA+1 y BAJA+2 pero solo un pequeño porcentaje (10% está propuesto) de los CONTINUA
- genera una salida con las probabilidades de los mejores modelos que se prueban en la Optimización Bayesiana, para utilizar luego esas probabilidades en un stacking.

Donde entrenar/tetear y cuanto undersampling de los CONTINUA se realiza es controlado al comienzo del script en las siguientes lineas:

```
ktest_mes_hasta  <- 202011  #Esto es lo que uso para testing
ktest_mes_desde  <- 202011


ktrain_subsampling  <- 0.1   #el undersampling que voy a hacer de los continua

ktrain_mes_hasta    <- 202010  #Obviamente, solo puedo entrenar hasta 202011
ktrain_mes_desde    <- 202001
ktrain_meses_malos  <- c( 202006 )  #meses que quiero excluir del entrenamiento

kgen_mes_hasta    <- 202011   #La generacion final sin undersampling
kgen_mes_desde    <- 202001
```

ktrain_subsampling <- 0.1 indica con que parte de los CONTIUA se va a trabajar; 0.1 indica el 10% de los CONTINUA. Con un subsampling más pequeño el dataset tendrá menos registros, la generación del modelo será más rápida, PERO la calidad del modelo disminuirá.

Si usted está angustiado y temeroso con la idea del undersampling de los CONTINUA y para estar traquilo con su conciencia quiere volver a trabajar con todos los datos "para darle la oportunidad al modelo aprenda lo más posible" simplemente cambie a `ktrain_subsampling <- 1` y el script trabajará con todos los datos.

A pesar que la Optimización Bayesiana se realizará sobre el dataset con menos registros en caso de elegir un undersampling menor a 1, y su objetivo es apenas devolver los mejores hiperparámetros, para la generación del modelo final se utiliza el dataset completo. Recuerde que en la carpeta Kaggle se guardan las salidas superadoras de la Optimización Bayesiana.

Ahora pasamos a la parte en donde no utilizamos más el 5-fold cross validation.
Se hará simplemente un training y testing.
En el ejemplo mostrado, se entrena en los ocho de
{ 202001, 202002, 202003, 202004, 202005, 202007, 20208, 202009 }
notar que en este caso se está excluyendo el mes 202006 , decisión tomada debido a la muy mala calidad de los datos de gran parte de las variables de ese mes.
Donde se entrena se indica en estas tres lineas

```
ktrain_mes_hasta    <- 202010  #Obviamente, solo puedo entrenar hasta 202011
ktrain_mes_desde    <- 202001
ktrain_meses_malos  <- c( 202006 )  #meses que quiero excluir del entrenamiento
```

Ahora pasamos al donde se testea. En el ejemplo presentado, se está testeando en
{ 202011 }
Recordar que 202011 es el último mes con la clase completa, es el último mes donde puedo testear (o entrenar). Decididamente no es posible utilizar 202012 y menos aún 202101.
Donde se testea se indica en las lineas

```
ktest_mes_hasta  <- 202011  #Esto es lo que uso para testing
ktest_mes_desde  <- 202011
```

Usted notará que es posible testear en varios meses, pero en este caso se decidió entrenar en un solo mes.

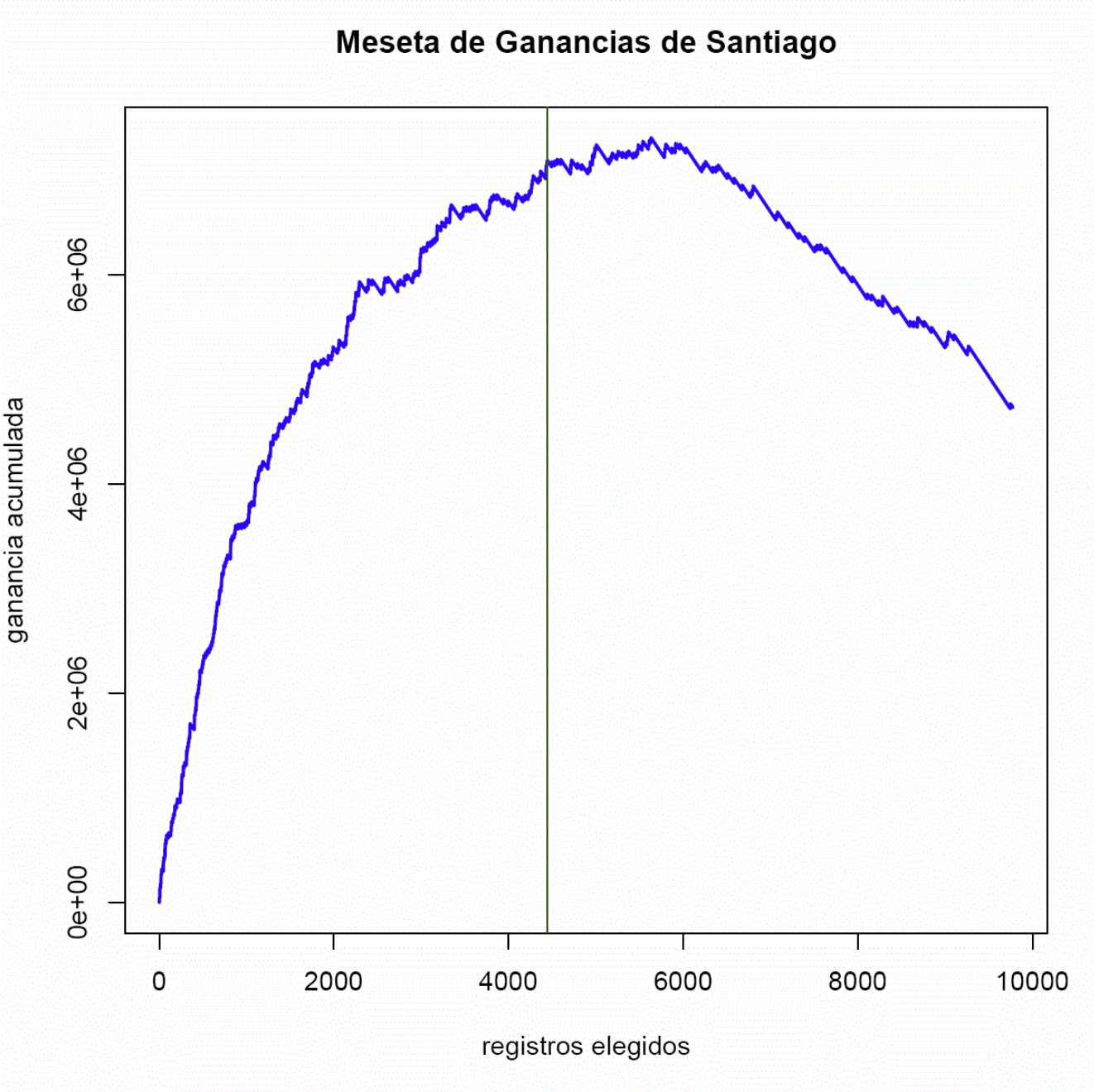
Es muy recomendable que suceden estas dos cosas al mismo tiempo:

- Se entrene en meses anteriores a donde se testea.
- No se testee en meses que se usan para entrenar.

Finalmente, cada vez que se genera un modelo para Kaggle, se entrena SIN undersampling de los CONTINUA, para que el modelo sea lo más preciso posible. Además uno quisiera que en ese entrenamiento se utilicen los meses más recientes, recordar que 202011 se está utilizando para testear y no para entrenar, ahora se quiere utilizar 202011 para ese entrenamiento. Esto se controla con estas lineas

```
kgen_mes_hasta    <- 202011   #La generacion final sin undersampling
kgen_mes_desde    <- 202001
```

Este script al igual que los anteriores de Optimización Bayesiana genera en la carpeta Kaggle distintas salidas, una por cada modelo superador que va en econtrando durante la bayesiana. Pero, este script agrega una carpeta llamada `meseta` en donde genera varias nuevas salidas para Kaggle, cada una enviando distinta cantidad de registros. Investigue estos archivos, pruébelos, suba algunos a Kaggle y observe cuanto dan en el Public Leaderboard. Esto está relacionado con lo visto en la clase del 01-octubre y los siguientes gráficos:



Cada corrida de este script generá un nuevo tipo de archivo por primera vez, uno que irá a la carpeta `modelitos` del Bucket de Google Cloud.
Ese archivo tiene para cada modelo generado para Kaggle probabilidades que devuelve el modelo si es aplicado a *todo* el dataset.

Un ejemplo de las primeras lineas de estos archivos es:

num_cliente	foto_mes	E1311_1	E1311_5	E1311_6	E1311_10	E1311_12	E1311_34
4572266	201804	0.00046	0.00022	0.00020	0.00009	0.00041	0.00022
4572266	201805	0.00026	0.00036	0.00039	0.00014	0.00023	0.00003
4572266	201806	0.00090	0.00076	0.00027	0.00026	0.00033	0.00007
4572266	201807	0.00018	0.00028	0.00031	0.00009	0.00009	0.00005
4572266	201808	0.00048	0.00037	0.00058	0.00040	0.00090	0.00009

El experimento corrido fue el 1311
La optimización bayesiana tuvo mejoras en las iteraciones 1, 5, 6, 10, 12 y 34
La columna `E1311_1` son las probabilidades que devolvió el modelo 1 cuando se aplica a todo el dataset. Cada columna es el resultado de un modelo.
Cada experimento tendrá sus columnas.
La idea es con todas las columnas devueltas por todos los experimentos generar un nuevo dataset, un dataset de probabilidades, y sobre ese datset entrenar un modelo de segundo nivel, lo que se conoce como Stacking.

A esta altura usted se estará preguntando "¿Cómo se generaron las probabilidades para todos los registros del dataset? ¿Se aplicó el modelo en los mismos datos que se entrenó?" pues no, se ha tenido el cuiado de hacer más de un modelo con los mismos hiperparámetros, y aplicarlo a los datos que no fueron utilizados para entrenar.

Obviamente la cantidad de memoria RAM que necesita este script para correr depende del tamaño del dataset a procesar, tamaño que usted controla según las palancas que puso en TRUE al correr el script `811_dataset_epic.r` . Si usted cayó un un frenesí de activar demasiadas palancas su dataset tendrá miles de columnas y muy posiblemenente ni siquiera 256 GB de memoria RAM le sean suficientes.

Cuantas más corridas usted realice del 821, mas modelitos tendrá y un mejor stacking podrá realizar.

Quizás suene repetitivo, pero es muy importante notar que el objetivo del undersampling del 10% de los CONTINUA tiene como objetivo acelerar la corrida, y no el balanceo de las clases. LightGBM es lo suficientemente robusto como para trabajar con clases desbalanceadas, no como sus ancestros.

6. Script `834_dataset_stacking.r`

Este script va a la carpeta `modelitos`, hace el merge de TODOS los archivos que se encuentran en esa carpeta, y finalmente le agrega la clase.

Cada vez que usted corra este script deberá cambiar la linea y actualizar la versión.

```
version  <- "v002"  #cambiar cada vez, asi se tiene versionado del dataset
```

Este script genera como salida un dataset, que sun la versión se llama similar a

```
./datasets/dataset_stacking_v002.csv.gz
```

No tema que alguno de los modelos generados sea malo, ya que en la medida que usted utilizó para testing meses posteriores a donde entrenó (recomendación dada en el paso 5 de esta Tarea para el Hogar) , si un modelito no es bueno, el modelo de segundo nivel no utilizará esa columna.

Cuantas más corridas usted realice del 821, mas modelitos tendrá y un mejor stacking podrá realizar.

7. `845_stacking.r`

Este script es exactamente el mismo que el 822, solo que para claridad conceptual se lo renombró y se lo invoca con el dataset generado por el script `834_dataset_stacking.r`

Este script genera un modelo de segundo nivel ya que trabaja sobre alguno de los datasets que genera cada corrida del script `834_dataset_stacking.r`

La entrada es el dataset está en linea

```
karch_dataset     <- "./datasets/dataset_stacking_v002.csv.gz"    #este dataset se genero en el script 811_dataset_epic.r
```

se debe cambiar según el dataset (de stacking) que se desee utilizar.

Enjoy the Power of Stacking !