TD VI: Inteligencia Artificial
Trabajo práctico 1 (2023 2do semestre)

Objetivo

Ya vimos cómo funcionan los árboles de decisión. También vimos la importancia de validar modelos, pues no sabemos de antemano qué *performance* tendrán los mismos en producción (pero, si hacemos las cosas bien, la podemos estimar). En este trabajo práctico (TP), vamos a ver cómo distintas decisiones de modelado impactan sobre la performance de los modelos (estimada mediante *repeated validation set*). Para ello, seguiremos un enfoque experimental.

Concretamente, la idea será adquirir intuiciones referidas a cómo distintas estrategias de manipulación de datos y distintas características de los mismos impactan sobre la capacidad predictiva de los modelos (árboles de decisión en este caso). Para ello, deberán llevar adelante distintos experimentos.

Ejemplo de un experimento

A continuación, se presentan resultados parciales de un experimento modelo. Los mismos se obtienen a partir de los dos *scripts* provistos (`provided_functions.R` y `sample_exp.R`) y utilizan los conjuntos de datos provistos en este TP (`customer_churn.csv`[1] y `heart.csv`[2]).[3] Concretamente, al ejecutar el script `sample_exp.R` se lleva adelante un experimento que hace lo siguiente:

- Carga los dos datasets provistos.
- Luego, la función `run_experiment` calcula, para cada dataset, la performance que tendrán árboles entrenados con profundidad máxima de 1 a 30. Esto lo hace para cuatro variantes de los datos (los cuales se modifican asignando distintos valores de `prop_NAs` e `impute_NAs` en `preprocess_control` — el objeto que toma como *input* la función `preprocess_data`). Concretamente, las cuatro variantes son las siguientes:

    1) Los datos **tal cual** son, dejando que el propio árbol se haga cargo de eventuales valores faltantes (`NA`s en R). Valor de `prop_NAs = 0` y valor de `impute_NAs` igual a `FALSE`.
    2) Los datos **tal cual** son; **pero**, para cada variable, **imputando** los valores faltantes numéricos con la media de la variable (medida sobre entrenamiento, como debe ser para evitar *data leakage*[4]) y con la moda en caso de la variable en cuestión ser categórica (también calculada sobre entrenamiento). Valor de `prop_NAs = 0` y valor de `impute_NAs` igual a `TRUE`.
    3) Los datos perturbados de manera tal que, en cada variable predictora, se reemplaza un **70%** de los valores por valores ***missing*** (`NA`s en R), dejando que el propio árbol se haga cargo de eventuales valores faltantes. Valor de `prop_NAs = 0.7` y valor de `impute_NAs` igual a `FALSE`.
    4) Los datos perturbados de manera tal que, en cada variable predictora, se reemplaza un **70%** de los valores por valores **missing**, **pero imputando** con media y moda como se mencionó arriba. Valor de `prop_NAs = 0.7` y valor de `impute_NAs` igual a `TRUE`.

- `run_experiment` también guarda los resultados del experimento en un archivo de texto plano (`exp_sample.txt`). Este archivo contiene detalles de cada modelo entrenado como, por ejemplo, la performance obtenida en cada repetición de la validación y la profundidad efectiva de cada árbol (que

[1] https://archive.ics.uci.edu/dataset/563/iranian+churn+dataset
[2] https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset
[3] Los scripts se encuentran documentados en el apéndice de estas consignas.
[4] El tema *data leakage* lo veremos en próximas teóricas.

puede ser menor a la máxima permitida). Noten que el archivo consta de 7.200 registros (es decir, ¡el script entrena 7.200 árboles!).[5]

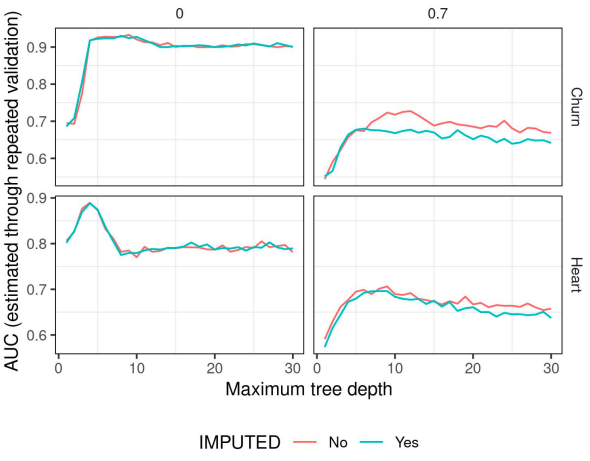
- Finalmente, al ejecutarse la función `plot_exp_results`, se cargan dichos datos, se estima el valor de AUC de cada configuración (i.e., para cada configuración, se promedian los valores de AUC de validación de las distintas repeticiones), se crea un gráfico con `ggplot` y se guarda al mismo en formato JPG.

Detalles sobre el experimento:

- La performance la medimos mediante una métrica que se conoce como área bajo la curva ROC (ROC por *receiver operating characteristic*) o, simplemente, AUC. En las siguientes clases teóricas, veremos en detalle qué captura la misma. Por lo pronto, a los fines de este TP, alcanza con saber que un valor igual a 1 indica que el modelo predice perfectamente, mientras que un valor de 0,5 indica que el modelo no logra predecir nada útil (siendo esta la peor de las situaciones).[6] Utilicen AUC como la métrica de performance predictiva de los modelos a lo largo de todo este TP.
- Correr este experimento, y más aún los que ustedes deberán llevar a cabo, implica entrenar miles de modelos. Para poder hacerlo en un tiempo razonable, el código paraleliza la construcción de los árboles. Sin embargo, dado que trabajar en paralelo dificulta mucho *debuggear* el código, también se puede desactivar la opción de paralelizar la construcción de los árboles asignando la variable `PARALLELIZE` a `FALSE`.
- No es necesario correr el experimento siempre que se quieran analizar los resultados. Si ustedes ya corrieron el experimento, pueden utilizar los resultados guardados, haciendo que `RERUN_EXP` valga `FALSE`.
- Se presentan los resultados del experimento en una única figura. Aunque consideraremos como válido presentar los resultados de los experimentos con una única **buena** figura, se valorará que incorporen alguna(s) más (tampoco un número exagerado) o que presenten tablas pertinentes que contengan patrones interesantes encontrados.

A continuación, se muestra la figura que genera el script `sample_exp.R` (el título no forma parte de la figura).

Figura 1. Valores de AUC obtenidos al seguir distintas estrategias de imputación de valores faltantes



[5] El valor de 7.200 sale de hacer la siguiente cuenta: cantidad de datasets para los cuales se predice * cantidad de valores posibles para `impute` * cantidad de valores posibles para `prop_NAs` * máxima profundidad permitida * cantidad solicitada de repeticiones de validación. Máxima profundidad permitida es `max_maxdepth` y cantidad solicitada de repeticiones es `val_reps`. Entonces, 2 * 2 * 2 * 30 * 30 = 7.200.

[6] Valores de AUC menores a 0,5 no debieran ocurrir; pero, si ocurriesen, podrían ser aprovechados para predecir.

Noten que la figura está compuesta por distintos paneles (facetas/*facets*). Dentro de cada facet, se ve la performance de validación obtenida para distintas profundidades máximas, diferenciando por si los NAs fueron imputados (en verde agua, ≈ azul) o no (en coral, ≈ rojo). A su vez, cada facet permite ver los resultados para diferentes datasets (fila del facet) o porcentaje de NAs generados (columna del facet). A modo de ejemplo, el facet en la fila Churn y columna 0.7 muestra los resultados obtenidos para el conjunto de datos Churn cuando el 70% de los valores de cada columna son pasados a NAs.

Parte muy importante de la evaluación de este TP será **interpretar los resultados** obtenidos en figuras como la presentada arriba. El análisis puede centrarse en reportar 1) patrones generales (e.g., a medida que mayor es el porcentaje de NAs, menor es la performance de los modelos), en reportar 2) patrones particulares observados en un dataset y, posiblemente, no en otros (e.g., se observa que imputar NAs afecta la performance de manera negativa en mayor medida en el conjunto de datos Churn que en el de Heart), o en reportar 3) cualquier patrón o particularidad que ustedes quieran destacar. Además, se valorará que, para los patrones reportados, se **postulen hipótesis** de por qué ocurren (basadas, por ejemplo, en conceptos teóricos de Inteligencia Artificial o en conocimiento sobre el dominio de los datos en cuestión). En caso de que para alguno(s) de los patrones reportados no encuentren ninguna hipótesis razonable (cosa que es posible), indiquen que no pudieron encontrar una hipótesis convincente para la ocurrencia del patrón en cuestión.

Para hacer

1) **Selección de un conjunto de datos adicional.** Busquen y elijan un dataset adicional para analizar. El mismo debe tener las siguientes características:

    a) Debe poder utilizarse para atacar un problema de clasificación binaria (sí o sí). Noten que ustedes pueden construir una variable binaria a partir de una variable numérica (e.g., creando una variable que indique si el valor de la variable numérica es mayor o igual a algún otro valor fijo). También, a partir de una variable categórica con más de dos valores se puede construir una variable binaria (e.g., preguntando si el valor es igual a una categoría dada o no). Estas modificaciones pueden ser hechas por ustedes.
    b) Sí o sí debe tener atributos predictores numéricos. Idealmente, al menos 4 predictores deberían ser numéricos. Eso sí, puede no tener atributos predictores categóricos.
    c) Los atributos categóricos no deben estar representados como números. A modo de ejemplo, en el conjunto de datos de *churn* provisto, los valores de `complains`, `tariff_plan`, `status` y `churn` estaban *encodeados* mediante números enteros. Nosotros modificamos el archivo CSV para que en su lugar aparezcan las categorías como palabras y no como números. Ustedes deberán hacer lo mismo con sus dataset.
    d) El conjunto de datos no debe superar las 10.000 observaciones y no debe tener ni un número ínfimo de predictores (e.g., 2) ni un número excesivo (e.g., miles). Si ustedes quisieran analizar un conjunto de datos más grande, les pedimos que hagan un muestreo aleatorio del mismo para que no tenga más de 10.000 observaciones.
    e) La variable objetivo no debe ser trivial de predecir. La idea es que, para predecir la variable, debe ser necesario entrenar un modelo complejo como un árbol de decisión, y que no baste con algo como una simple regla.

    Deberán entregar lo siguiente:

    Una pequeña notebook en formato RMarkdown (tutorial) que:

1) Indique precisamente de dónde se obtuvieron los datos. (A modo de ilustración, que el enlace adjunto no sea https://data.buenosaires.gob.ar/dataset/, sino que sea https://data.buenosaires.gob.ar/dataset/traslados-covid-19).
2) Presente las características principales del conjunto de datos (e.g., cantidad de observaciones, cantidad de variables, tipo de variables, presencia de valores faltantes).
3) Explique qué es lo que se intentará predecir.
4) Haga todas las transformaciones a los datos requeridas.
5) Guarde el archivo a analizar en formato CSV tradicional (texto plano delimitado por comas, en donde la primera fila debe contener los nombres de las variables).

Esta notebook debe titularse `ej_1` y debe ser entregada tanto en formato Rmd (`ej_1.Rmd`) como en formato PDF (`ej_1.pdf`).

En caso de no saber por dónde empezar a buscar el dataset adicional, a continuación se brindan enlaces a algunas páginas que proveen bases de datos. Tengan en cuenta que algunas de las librerías vistas en clase también proveen datasets "de juguete". Aclaraciones: 1) dichas bases no necesariamente cumplen los requisitos indicados (el chequeo corre por su cuenta) y 2) el dataset adicional tranquilamente puede obtenerse de algún otro lugar.

- https://data.buenosaires.gob.ar/dataset/
- https://datos.gob.ar/
- https://www.gapminder.org/data/
- https://github.com/owid
- https://dataverse.harvard.edu/
- https://www.kaggle.com/datasets
- https://archive.ics.uci.edu/

2) **Experimentos sugeridos**. Elijan y lleven adelante 2 de los 4 experimentos sugeridos a continuación.

MUY IMPORTANTE: para que sea posible replicar sus resultados, al realizar cada uno de los experimentos (tanto los de este ejercicio como el del ejercicio 3), en todos los scripts entregados, fijen una semilla (`set.seed()`) concatenando los 3 últimos dígitos del DNI de los tres miembros del grupo, ordenados en función del apellido alfabéticamente. Por ejemplo, si los miembros del grupo son Ariel Martínez (DNI: 10.234.567), Lucía Gómez (DNI: 89.102.345) y Carlos González (DNI: 67.891.023), la semilla para fijar por este grupo debe ser `345023567` (primero Gómez con 345, luego González con 023 y por último Martínez con 567), quedando pues `set.seed(345023567)`.

Opción 1. Se pide que lleven adelante un experimento similar al presentado como ejemplo, pero con dos diferencias:

1) En lugar de probar valores de `prop_NAs` iguales a 0 y 0,7, prueben valores de 0,1 a 0,9 en saltos de 0,2 (es decir, lo que se obtiene en R al ejecutar `seq(0.1, 0.9, 0.2)`).
2) Además de analizar los dos datasets provistos, incorporen también al análisis el dataset que ustedes eligieron y confeccionaron en el punto 1 de este TP.

Noten que, en términos de programación, para llevar adelante lo que se pide, prácticamente no debe modificarse lo provisto en `sample_exp.R`.

En caso de elegir esta opción, deberán entregar lo siguiente:

a) Un script con nombre **`exp_1.R`** que lleve adelante el experimento y que genere todas las tablas y gráficos que luego se analizarán (algo equivalente a **`sample_exp.R`** pero modificado por ustedes).
b) Una notebook de RMarkdown en donde se analice el impacto que tienen los NAs sobre las predicciones, a través de los distintos datasets, y en donde se estudie qué tan efectivas son las distintas maneras de lidiar con los NAs (imputar media y moda versus dejar que el árbol los trabaje). Recuerden que se valorará que, para los patrones reportados, se postulen hipótesis de por qué ocurren. En caso de que para alguno(s) de los patrones reportados no encuentren ninguna, indiquen que no pudieron encontrar una hipótesis convincente para la ocurrencia del patrón en cuestión. **IMPORTANTE**: esta notebook debe llamar al script **`exp_1.R`** al comienzo de la misma (e.g., **`source("exp_1.R")`**). La notebook debe titularse **`exp_1`** y debe ser entregada tanto en formato Rmd (**`exp_1.Rmd`**) como en formato PDF (**`exp_1.pdf`**).

Opción 2. En este experimento, estudiarán qué impacto tiene representar las variables categóricas mediante one-hot-encoding (OHE)[7] en lugar de dejar que el árbol las trabaje a su manera. A su vez, al hacer OHE, existe la opción de hacer que toda la fila correspondiente a una variable con NA valga también NA o de crear una columna indicadora de NAs. Véase el ejemplo abajo:

```
> library(caret)
>
> main_df <- data.frame(var_1 = factor(c("A", NA, "B")),
+                       var_2 = factor(c("C", "D", NA)))
> print(main_df)
  var_1 var_2
1     A     C
2  <NA>     D
3     B  <NA>
>
> # Opción 1: toda la fila correspondiente a la variable como NAs
> ohe <- dummyVars(~., main_df)
> print(predict(ohe, main_df))
  var_1.A var_1.B var_2.C var_2.D
1       1       0       1       0
2      NA      NA       0       1
3       0       1      NA      NA
>
> # Opción 2: nueva columna para los NAs
> main_df$var_1 <- addNA(main_df$var_1)
> main_df$var_2 <- addNA(main_df$var_2)
> ohe <- dummyVars(~., main_df)
> print(predict(ohe, main_df))
  var_1.A var_1.B var_1.NA var_2.C var_2.D var_2.NA
1       1       0        0       1       0        0
2       0       0        1       0       1        0
3       0       1        0       0       0        1
```

Se pide que en el experimento se analicen los efectos de las distintas maneras propuestas de trabajar variables categóricas, considerando a su vez valores de **`prop_NAs`** iguales a 0; 0,25; 0,5 y 0,75 (sólo estos cuatro valores). **IMPORTANTE**: noten que **`preprocess_control`** puede configurarse para hacer todo lo que se pide:

- El caso de dejar que el árbol trabaje las categóricas es idéntico a uno de los casos del experimento 1.
- El caso de hacer OHE sin crear una columna para los NAs se consigue con **`treat_NAs_as_new_levels`** igual a **`FALSE`** y **`do_ohe`** igual a **`TRUE`**.

[7] https://miro.medium.com/v2/resize:fit:1358/1*ggtP4a5YaRx6l09KQaYOnw.png

- El caso de OHE sí creando una columna para los NAs se consigue con `treat_NAs_as_new_levels` igual a `TRUE` y `do_ohe` igual a `TRUE`.

MUY IMPORTANTE: dado que sus conjuntos de datos pueden no tener variables categóricas (algo que invalidaría este experimento), se pide que a lo largo de todo este experimento el valor de `discretize` en las tres configuraciones de `preprocess_control` que usen sea siempre igual a `TRUE` (y el valor de `n_bins` sea siempre igual a 10). Esto hará que todas las variables continuas sean discretizadas (transformadas en categóricas), reemplazando a la variable continua por una nueva variable categórica con 10 intervalos de valores distintos, cada uno con igual cantidad de observaciones.[8]

En caso de elegir esta opción, deberán entregar lo siguiente:

a) Un script con nombre `exp_2.R` que lleve adelante el experimento y que genere todas las tablas y gráficos que luego se analizarán (algo equivalente a `sample_exp.R` pero modificado por ustedes).
b) Una notebook de RMarkdown en donde se analice el impacto que tienen las distintas maneras de trabajar atributos categóricos y cómo esto se ve afectado por la presencia de NAs. Recuerden que se valorará que, para los patrones reportados, se postulen hipótesis de por qué ocurren. En caso de que para alguno(s) de los patrones reportados no encuentren ninguna, indiquen que no pudieron encontrar una hipótesis convincente para la ocurrencia del patrón en cuestión. **IMPORTANTE**: esta notebook debe llamar al script `exp_2.R` al comienzo de la misma (e.g., `source("exp_2.R")`). La notebook debe titularse `exp_2` y debe ser entregada tanto en formato Rmd (`exp_2.Rmd`) como en formato PDF (`exp_2.pdf`).

Opción 3. Las variables categóricas pueden ser nominales (sin un orden innato; e.g., una variable que tome los valores azul, rojo o amarillo) u ordinales (con un orden innato; e.g., una variable que tome los valores malo, bueno, excelente). Al momento de hacer OHE, si uno trabaja las variables categóricas ordinales como si fueran nominales, se pierde el orden de las mismas. Por este motivo es que se suele recomendar transformar las variables categóricas ordinales a numéricas de manera que se conserve el orden (e.g., malo=0, bueno=1, excelente=2). En este experimento, evaluarán esa recomendación aprovechando que, al discretizar variables numéricas con la opción `discretize` igual a `TRUE`, en `preprocess_control` se crean variables categóricas ordinales.

Puntualmente, ustedes deberán llevar adelante un experimento en donde se estudie, para cada uno de los tres datasets analizados, cómo impacta tratar las variables categóricas ordinales con OHE estándar (algo que ya se propone en el experimento anterior; i.e., `discretize` igual a `TRUE`, `n_bins` igual a 10 y `ord_to_numeric` igual a `FALSE` en `preprocess_control`), versus convertirlas en un número que refleje el orden de las categorías (algo que se logra con `discretize` igual a `TRUE`, `n_bins` igual a 10 y `ord_to_numeric` igual a `TRUE` en `preprocess_control`). En este experimento, se pide que NO prueben con valores de `prop_NAs` mayores a 0 y que no imputen NAs (que el mismo árbol los trabaje; es decir, que `prop_NAs` sea siempre igual a 0).

En caso de elegir esta opción, deberán entregar lo siguiente:

[8] "Equal-frequency discretization sorts the continuous variable into intervals with the same number of observations. The interval width is determined by quantiles. Equal-frequency discretization is particularly useful for skewed variables as it spreads the observations over the different bins equally" (fuente).

a) Un script con nombre `exp_3.R` que lleve adelante el experimento y que genere todas las tablas y gráficos que luego se analizarán (algo equivalente a `sample_exp.R` pero modificado por ustedes).
b) Una notebook de RMarkdown en donde se analice el impacto que tienen las distintas maneras de trabajar atributos categóricos ordinales. Recuerden que se valorará que, para los patrones reportados, se postulen hipótesis de por qué ocurren. En caso de que para alguno(s) de los patrones reportados no encuentren ninguna, indiquen que no pudieron encontrar una hipótesis convincente para la ocurrencia del patrón en cuestión. **IMPORTANTE**: esta notebook debe llamar al script `exp_3.R` al comienzo de la misma (e.g., `source("exp_3.R")`). La notebook debe titularse `exp_3` y debe ser entregada tanto en formato Rmd (`exp_3.Rmd`) como en formato PDF (`exp_3.pdf`).

Opción 4. Este experimento es diferente a los demás. Estudiarán cómo se comportan los árboles de decisión ante ruido en la variable a predecir (*label noise*). Concretamente, verán cómo las *noisy labels* afectan la performance de los árboles de decisión. Noten que el código entregado permite crear ruido en las labels. Concretamente, al fijar el valor de `prop_switch_y` a un valor mayor a 0 dentro de `preprocess_control`, la función `preprocess_data` modificará las labels del dataset de entrenamiento (no del de validación) de la siguiente manera:

a) Seleccionará un `prop_switch_y`% de las labels de entrenamiento al azar.
b) Para estas labels muestreadas, *switchea* su valor (es decir, si valía 1 pasa a valer 0 y si valía 0 pasa a valer 1).

Tener ruido en las labels es algo común, de modo que este experimento es muy importante. Concretamente, se pide que lleven adelante un experimento que, para valores de `prop_switch_y`% que van de 0 a 0,5 en saltos de 0,025 (lo que se obtiene en R al ejecutar `seq(0, 0.5, 0.025)`), permita estudiar cuál es la máxima performance que se puede obtener con árboles de decisión. Se pide que, en este caso, no reporten valores de AUC para las 30 profundidades diferentes sino que reporten, para esas 30 profundidades probadas, cuál fue el mayor AUC obtenido. También, se pide que no modifiquen ningún otro hiperparámetro de entrenamiento de los árboles, salvo la profundidad.

En caso de elegir esta opción, deberán entregar lo siguiente:

a) Un script con nombre `exp_4.R` que lleve adelante el experimento y que genere todas las tablas y gráficos que luego se analizarán (algo equivalente a `sample_exp.R` pero modificado por ustedes).
b) Una notebook de RMarkdown en donde se analice el impacto que tienen las noisy labels. Recuerden que se valorará que, para los patrones reportados, se postulen hipótesis de por qué ocurren. En caso de que para alguno(s) de los patrones reportados no encuentren ninguna, indiquen que no pudieron encontrar una hipótesis convincente para la ocurrencia del patrón en cuestión. **IMPORTANTE**: esta notebook debe llamar al script `exp_4.R` al comienzo de la misma (e.g., `source("exp_4.R")`). La notebook debe titularse `exp_4` y debe ser entregada tanto en formato Rmd (`exp_4.Rmd`) como en formato PDF (`exp_4.pdf`).

3) **Experimento propio**. Ahora es momento de que propongan y lleven adelante ustedes un experimento. Recuerden que la idea de este TP es adquirir intuiciones referidas a cómo distintas estrategias de manipulación de datos y distintas características de los mismos impactan sobre la capacidad predictiva de los modelos (árboles de decisión en este caso). Por lo tanto, el experimento propuesto tiene que ser sobre alguna estrategia de manipulación de datos; no, por ejemplo, sobre los hiperparámetros del modelo. A modo de guía, potenciales propuestas podrían estar vinculadas a

agregar ruido, en vez de a la variable objetivo como en la opción 4, a los atributos (por ejemplo, sólo al atributo más importante para el árbol o a X% de los atributos disponibles) o modificar la variable objetivo para poder comparar clases balanceadas versus clases desbalanceadas. Aclaración: la propuesta no necesariamente tiene que ser una de las recién mencionadas.

Para este experimento, si lo requieren, pueden generar una nueva versión del script `provided_functions.R` y modificarlo a gusto (llámenlo `provided_functions_exp_propio.R`).

Deberán entregar lo siguiente:

a) Un script con nombre `exp_propio.R` que lleve adelante el experimento y que genere todas las tablas y gráficos que luego se analizarán (algo equivalente a `sample_exp.R` pero modificado por ustedes).
b) En caso de modificar el script `provided_functions.R`, deberán entregar `provided_functions_exp_propio.R`.
c) Una notebook de RMarkdown en donde motiven el experimento y detallen qué patrones interesantes o no interesantes encontraron. Recuerden que para cada patrón que reporten deben plantear una hipótesis de por qué ocurre. Recuerden que se valorará que, para los patrones reportados, se postulen hipótesis de por qué ocurren. En caso de que para alguno(s) de los patrones reportados no encuentren ninguna, indiquen que no pudieron encontrar una hipótesis convincente para la ocurrencia del patrón en cuestión. **IMPORTANTE**: esta notebook debe llamar al script `exp_propio.R` al comienzo de la misma (e.g., `source("exp_propio.R")`). La notebook debe titularse `exp_propio` y debe ser entregada tanto en formato Rmd (`exp_propio.Rmd`) como en formato PDF (`exp_propio.pdf`).

Modalidad de entrega

- Tal como ya dijimos, el trabajo se debe realizar en **grupos de tres personas** (con 1 o 2 excepciones en la sección 2, según si se hace 1 grupo de 4 o 2 de 2).
- La fecha límite de entrega es el **martes 22 de agosto a las 23:59 h**.
- Recomendación: no lo dejen para último momento; son varias las cuestiones pedidas. La segunda parte de la clase **práctica del lunes 14** de agosto la vamos a destinar a que cada grupo pueda avanzar en la resolución de este TP y consultar dudas al respecto a los auxiliares docentes.
- Para realizar sus consultas sobre este TP, también cuentan con el foro llamado **TP1 | Consultas** en la página de la materia en el Campus Virtual. Todas las dudas que surjan en relación al TP1 envíenlas exclusivamente a este foro; no usen ningún otro. Esto debe ser así porque este es el foro que está configurado de forma que los mensajes enviados lleguen únicamente al cuerpo docente y a sus compañeros de grupo. En otras palabras, si un integrante de un grupo envía una pregunta por acá, tanto esa pregunta como la respuesta, luego dada por el cuerpo docente, podrán ser vistas sólo por los integrantes del grupo en cuestión.
- **Sólo 1** integrante del grupo debe realizar la **entrega**.
- Los archivos para entregar deben ponerse dentro de una carpeta llamada `tp1-gxx`, donde `xx` sea reemplazado por el número de grupo; por ejemplo, `tp1-g01`. La versión ZIP de esta carpeta debe subirse a la tarea llamada **TP1 | Entrega** en la página de la materia en el Campus Virtual. (Cuidado con los envíos a último minuto que la tarea se cierra a las 23:59:00).

- Deben entregar **11 ó 12 archivos** (recomendamos altamente revisar la siguiente *checklist* previo al cierre de la tarea, para asegurarse de que la carpeta antes mencionada contenga todos los archivos requeridos):

  - [ ] ~~`ej_1.Rmd`~~
  - [ ] ~~`ej_1.pdf`~~

  Si, en el ejercicio 2, eligieron la opción 1:
  - [ ] ~~`exp_1.R`~~
  - [ ] ~~`exp_1.Rmd`~~
  - [ ] ~~`exp_1.pdf`~~

  Si, en el ejercicio 2, eligieron la opción 2:
  - [ ] `exp_2.R`
  - [ ] `exp_2.Rmd`
  - [ ] `exp_2.pdf`

  Si, en el ejercicio 2, eligieron la opción 3:
  - [ ] `exp_3.R`
  - [ ] `exp_3.Rmd`
  - [ ] `exp_3.pdf`

  Si, en el ejercicio 2, eligieron la opción 4:
  - [ ] ~~`exp_4.R`~~
  - [ ] ~~`exp_4.Rmd`~~
  - [ ] ~~`exp_4.pdf`~~

  - [ ] ~~`exp_propio.R`~~
  - [ ] ~~`provided_functions_exp_propio.R`~~ (si es que modifican `provided_functions.R`)
  - [ ] ~~`exp_propio.Rmd`~~
  - [ ] ~~`exp_propio.pdf`~~

Anexo

Documentación de los scripts provistos

`provided_functions.R`

El script tiene como objetivo simplificar el proceso de preprocesamiento, modelado y evaluación de modelos de árboles de decisión, utilizando el lenguaje de programación R. Está compuesto por diversas funciones que, en conjunto, ofrecen un flujo de trabajo completo para cargar datos, realizar preprocesamiento, modelar y evaluar el desempeño de los árboles de decisión.

Descripción de las funciones

`load_df(filepath, dataset_name, var_to_predict)`

Carga datos desde un archivo CSV, realiza una validación básica de los datos y prepara un `data.frame` para su posterior análisis.

Parámetros:

- `filepath`: la ruta al archivo CSV.

- `dataset_name`: una cadena de caracteres que especifica el nombre del conjunto de datos.
- `var_to_predict`: una cadena de caracteres que especifica el nombre de la variable a predecir.

Retorno: una lista que contiene el `data.frame` preparado, el nombre del conjunto de datos y el nombre de la variable objetivo.

`preprocess_data(var_to_predict, train_df, val_df, prop_NAs, impute_NAs, treat_NAs_as_new_levels, do_ohe, discretize, n_bins, ord_to_numeric, prop_switch_y)`

Esta función realiza tareas de preprocesamiento en los `data.frames` de entrada, como el manejo de valores faltantes, la codificación one-hot, la imputación de valores faltantes, entre otras.

Parámetros:

- `var_to_predict`: el nombre de la variable objetivo a predecir.
- `train_df`: `data.frame` de entrenamiento.
- `val_df`: `data.frame` de validación.
- `prop_NAs`: proporción de valores faltantes para generar al azar en cada atributo.
- `impute_NAs`: valor lógico que indica si se deben imputar los valores faltantes.
- `treat_NAs_as_new_levels`: valor lógico que indica si se deben tratar los valores faltantes como nuevos niveles en las variables de tipo factor.
- `do_ohe`: valor lógico que indica si se debe realizar codificación one-hot.
- `discretize`: valor lógico que indica si se deben discretizar las variables continuas.
- `n_bins`: número de intervalos para discretizar variables numéricas (si `discretize` es `TRUE`).
- `ord_to_numeric`: valor lógico que indica si se deben convertir factores ordenados a valores numéricos (si `discretize` es `TRUE`).
- `prop_switch_y`: proporción de valores de la variable objetivo a intercambiar si se especifica.

Retorno: una lista que contiene los `data.frames` de entrenamiento y validación preprocesados.

`rep_val_estimate(var_to_predict, tree_control, data_df, prop_val, reps, preprocess_control)`

Esta función estima el AUC en la validación para un modelo de árbol de decisión. Realiza una validación repetida y calcula el AUC para cada repetición.

Parámetros:

- `var_to_predict`: el nombre de la variable objetivo a predecir.
- `tree_control`: parámetros de control para el árbol de decisión `rpart`.
- `data_df`: el `data.frame` que contiene los datos para el modelado.
- `prop_val`: proporción de observaciones para usar como validación.
- `reps`: número de repeticiones para la validación repetida.
- `preprocess_control`: una lista que contiene opciones de control para el preprocesamiento.

Retorno: un `data.frame` que contiene el AUC estimado para cada repetición, junto con la profundidad máxima utilizada en el árbol de decisión.

`est_auc_across_depths(data_to_pred, preprocess_control, max_maxdepth, prop_val, val_reps)`

Esta función estima el AUC para árboles de decisión de diferentes profundidades utilizando validación y paralelización. Realiza validación repetida para cada profundidad del árbol (de 1 a 30) y devuelve un `data.frame` con puntajes AUC.

Parámetros:

- `data_to_pred`: una lista que contiene el `data.frame`, el nombre de la variable objetivo y el nombre del conjunto de datos.
- `preprocess_control`: una lista que contiene opciones de control para el preprocesamiento.
- `max_maxdepth`: profundidad máxima del árbol a probar.
- `prop_val`: proporción de observaciones para usar como validación.
- `val_reps`: número de repeticiones para la validación repetida.

Retorno: un `data.frame` con puntajes AUC para diferentes profundidades del árbol.

`est_auc_across_depths_no_par(data_to_pred, preprocess_control, max_maxdepth, prop_val, val_reps)`

Esta función estima el AUC para árboles de decisión de diferentes profundidades, utilizando validación y paralelización. Realiza validación repetida para cada profundidad del árbol (de 1 a 30) y devuelve un `data.frame` con puntajes AUC. El motivo por el cual se entrega esta función es facilitar el *debugging* del código.

Parámetros:

- `data_to_pred`: una lista que contiene el `data.frame`, el nombre de la variable objetivo y el nombre del conjunto de datos.
- `preprocess_control`: una lista que contiene opciones de control para el preprocesamiento.
- `max_maxdepth`: profundidad máxima del árbol a probar.
- `prop_val`: proporción de observaciones para usar como validación.
- `val_reps`: número de repeticiones para la validación repetida.

Retorno: un `data.frame` con puntajes AUC para diferentes profundidades del árbol.

`sample_exp.R`

Este código en R tiene como objetivo realizar un experimento para evaluar el rendimiento de un modelo predictivo, bajo diferentes condiciones, utilizando árboles de decisión. El experimento implica la ejecución de varias combinaciones de conjuntos de datos, métodos de imputación y proporciones de valores faltantes. Además, el código incluye una función para representar gráficamente los resultados del experimento, utilizando la biblioteca ggplot2.

Bibliotecas requeridas

Asegúrense de cargar las siguientes bibliotecas, al inicio del script, para llevar a cabo el análisis de datos y la visualización:

- `ggplot2`: para la creación de gráficos. Tutorial.
- `dplyr`: para la manipulación de datos. Tutorial.

Variables globales

El código utiliza las siguientes variables globales para configurar opciones específicas:

- `PARALLELIZE`: indica si se debe paralelizar la computación.
- `N_THREADS`: número de hilos para el procesamiento paralelo.
- `N_BINS`: número de intervalos para la discretización, utilizada por la función de preprocesamiento de `data.frames`.
- `RERUN_EXP`: indica si se debe volver a ejecutar el experimento.

Funciones proporcionadas

Lean la documentación referida a las funciones proporcionadas (la anterior sección titulada `provided_functions.R`).

Funciones principales

`run_experiment(datasets_to_pred, filepath)`

Esta función ejecuta un experimento para evaluar el rendimiento de un modelo predictivo, bajo diferentes condiciones, y almacena los resultados en un archivo.

Parámetros:

- `datasets_to_pred`: una lista de `data.frames`, en donde cada uno contiene un conjunto de datos a predecir.
- `filepath`: la ruta al archivo donde se guardarán los resultados del experimento.

Esta función itera a través de diferentes combinaciones de conjuntos de datos, métodos de imputación y proporciones de valores faltantes. Para cada combinación, configura las opciones de preprocesamiento, realiza el experimento y almacena los resultados en una lista. Luego, combina los resultados en un único `data.frame` y lo guarda en el archivo especificado.

`plot_exp_results(filename_exp_results, filename_plot, width, height)`

Esta función representa gráficamente los resultados del experimento, utilizando la biblioteca ggplot2.

Parámetros:

- `filename_exp_results`: el nombre del archivo de resultados del experimento.
- `filename_plot`: el nombre del archivo para guardar el gráfico (por ejemplo, `mi_grafico.png`).
- `width`: el ancho del gráfico en pulgadas.
- `height`: la altura del gráfico en pulgadas.

Esta función lee los resultados del experimento, calcula los valores medios de AUC para diferentes condiciones experimentales y genera un gráfico de líneas utilizando ggplot2. El gráfico muestra los valores medios de AUC en función de la profundidad máxima del árbol, con líneas diferentes para diferentes métodos de imputación y *facetados* para diferentes conjuntos de datos y proporciones de valores faltantes. El gráfico resultante se guarda en el archivo especificado.