Checkpoint 3 : Ensambles de modelos

Preprocesamiento de datos

Para este tratamiento de los datos previo al entrenamiento de los modelos, contamos con lo realizado para el checkpoint 2, sin embargo realizamos varias modificaciones que permitieron mejorar el rendimiento de los modelos. Primeramente, para el tratamiento de *country* clusterizamos en dos categorías “PRT” y “NO-PRT” y lo pasamos a binario (adjunto en CHP1), debido a que realizar un label encoding no era beneficioso ya que los valores terminaban siendo bastante altos y no había una relación entre los datos que permita una escala numérica. Sin embargo, para lo que sí tenía sentido dicha transformación era para *arrival_date_month*, dondé si se podía establecer un orden. Por otro lado, un factor que habíamos ignorado, fue la estandarización de las variables discretas, esta normalización mejoró ampliamente el rendimiento de los modelos. Cabe destacar que nuevamente eliminamos los features *assigned_room_type* y *reserved_room_type*, ya que consideramos que la información condensada en esos valores se encuentra claramente representada en nuestra feature *missagned_room*. Finalmente, otro aspecto que tuvimos en cuenta, fue aplicar stratify para dividir el dataset entre train y test para tener una distribución parecida respecto el target.

Entrenamiento de los modelos

En este caso, para el entrenamiento de modelos tuvimos varias consideraciones globales. Primeramente, para la optimización por Cross Validation optamos siempre por una distribución de 10 folds, ya que aunque represente un 10% de los datos, nominalmente son aproximadamente 6.000 filas para testear, y ganamos el doble de validaciones cruzadas que con 5 folds, lo cual, a nuestro criterio, mejora el comportamiento de RandomizedSearch. En el caso de este último, en relación a las iteraciones fuimos cambiando entre 10 a 20, en relación a la performance del modelo a tratar. Este procedimiento lo aplicamos para la gran mayoría de los modelos, solamente exceptuando a Random Forest donde aplicamos un GridSearch.

En el caso de los ensambles, sobre todo en stacking, quisimos utilizar árboles con hiperparametros sacados del cv, sin embargo esto significó un volúmen de cómputo que excede lo esperado, hasta 4 horas tratando de ejecutar el modelo. Por eso optamos por árboles con hiper parámetros por default. Dada ésta mala experiencia, nos parece apropiado realizar una reducción de dimensionalidad del problema, dado que la gran cantidad de columnas (42) significa una alta exigencia al entrenamiento de modelos.

Conclusiones

Se utilizó el F1-score como medida de evaluación para determinar el mejor modelo de clasificación porque nuestro objetivo no era priorizar el recall o la precisión.
Es decir, para el tipo de análisis que se está realizando es igual de importante minimizar los falsos positivos como los falsos negativos.
Se encontró que el modelo de Random Forest con grid search evaluado con F1 tuvo el mejor desempeño en términos de esta métrica, lo que sugiere que el modelo fue capaz de encontrar un buen equilibrio entre la precisión y el recall teniendo un elevado porcentaje de aciertos.
Si bien se observó que el modelo de XGBoost tuvo un rendimiento comparable en términos de F1-score, su tiempo de ejecución fue mucho más largo, de hecho se intentó hacer un GridSearch pero su tiempo de ejecución fue demasiado largo, por eso se optó por un RandomizedSearch.
Luego modelos como el KNN con hiperparametros optimizados por CV y SVM optimizado a través de reducción de dimensionalidad o kernels no lograron tener un score comparable
En resumen, elegimos al modelo de Random Forest con grid search como nuestro mejor predictor ya que es el de mejor F1-score y además presentó una complejidad temporal razonable. Sin embargo quedó pendiente encontrar alguna forma de correr con mejor performance temporal otro tipo de modelos como XGBoost o un ensamble Stacking con un modelos con mejores hiperparametros, previamente encontrados por cross validation.