75.06 / 95.58

Organización de datos

Reporte TP N°1: Reservas de Hotel

Clasificación - Entrenamiento y Predicción

Checkpoint N°3: Ensambles

Alen Davies Leccese: 107084

Luca Mauricio Lazcano: 107044

Mayo 2023

1 Introducción

En esta tercer sección del trabajo, se continúan realizando entrenamientos y predicciones con distintos modelos. En esta ocasión, los modelos utilizados serán varios: K Nearest Neighbors (KNN), Support Vector Machine (SVM) y Random Forest (RF), provistos por Scikit-learn, y XGBoost. Además, se utilizará la técnica de ensamble, por Voting y por Stacking, que combina distintos modelos para aprovechar las mejores características de cada uno y obtener un mejor resultado.

Los datasets a utilizar serán el procesado en la primera etapa del trabajo, y los encodeados, que se utilizaron para la segunda etapa: `hotels_procesado.csv`, `hotels_test.csv`, `train_encoded` y `test_encoded`.

En primer lugar, se importan todas las librerías y herramientas a utilizar, las de graficación, los modelos de Scikit-learn y XGBoost, Pickle, y otras que resultarán útiles.

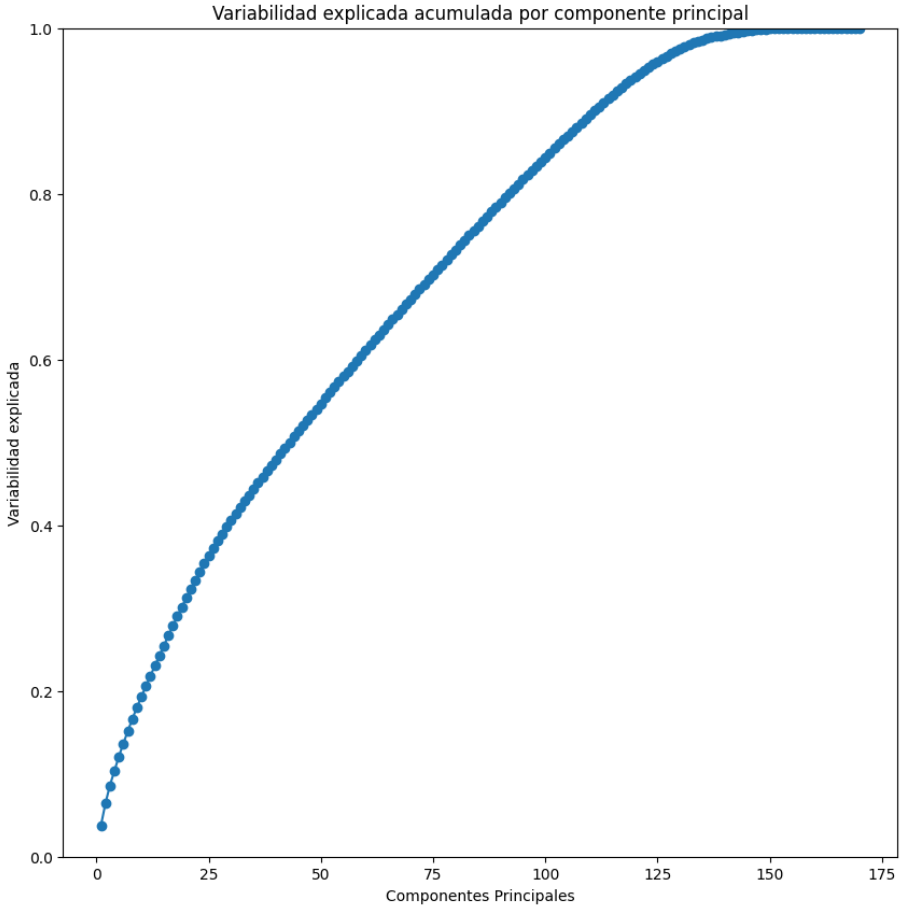
Pero antes de proceder con el entrenamiento de los modelos, recurrimos a una técnica para reducir la cantidad de datos, con el fin de reducir el tiempo de entrenamiento de los modelos,

sin perder demasiada información.

2 Reducción de la dimensionalidad por PCA

El análisis de componentes principales, o PCA ("principal components analysis"), es una técnica de la rama de la estadística, que se utiliza para describir a un conjunto de datos en términos de nuevas variables que tengan una mínima correlación entre sí. Es decir, busca las variables o datos que sean "redundantes" y los agrupa en un único componente. Esto permite reducir la dimensión del dataset, lo cual permite que los modelos tarden menos tiempo en entrenar, pero perdiendo una mínima precisión en el proceso.

Para este proceso, se utiliza la clase `PCA()` de `sklearn.decomposition`. En primer lugar, se normalizan los datos de prueba y entrenamiento con `StandardScaler()`. Luego, se transforman los datos con `PCA()` y se obtiene la "varianza explicada". Si lo visualizamos, se obtiene el siguiente gráfico de la acumulación de la varianza según el número de componentes principales.



Decidimos que una varianza del 85% era suficiente, por lo cual nos quedamos con los x componentes más importantes que acumulen dicha varianza. El número de componentos quedó en 102, para una varianza explicada del 85,48%.

Guardamos este modelo en un archivo `.pickle`.

3 K-Nearest neighbors (KNN)

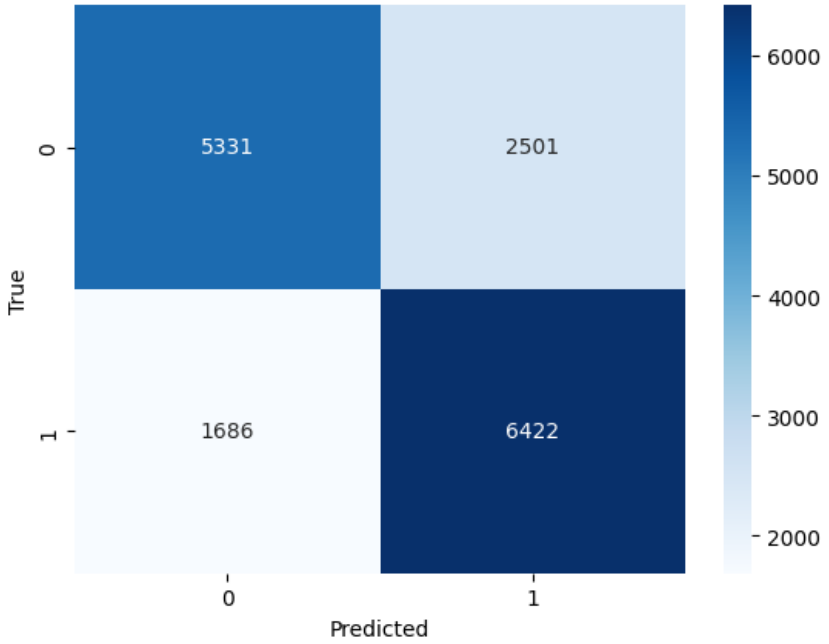
No ahondaremos demasiado en el detalle de cómo se crean y utilizan los modelos. Es esencialmente igual para todos, y ya fue detallado en el informe anterior.

En primer lugar se crea y entrena un modelo con los parámetros por defecto, para tener una idea aproximada y qué tanto hay que buscar mejorarlo.

Luego, para distintos número de "vecinos", graficamos qué tan bien aproxima el modelo. En el gráfico vemos que con una baja cantidad de neighbors el modelo aproxima mejor. Esto nos da una idea de qué rango de vecinos conviene explorar y permite reducirlo. Mediante cross-validation visualizamos mejor el F-1 score respecto a la cantidad de neighbors.

Para la búsqueda de hiperparámetros se realiza Random Search Cross Validation, con 5 folds, y se visualizan las métricas obtenidas. Con los parámetros obtenidos, concluimos que el modelo quedó "overfitteado".

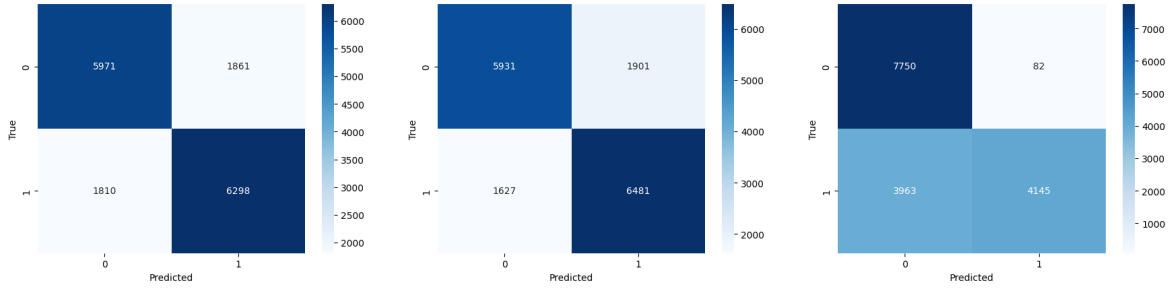
Se muestran las métricas, y se visualiza la matriz de confusión:



4 Support Vector Machine (SVM)

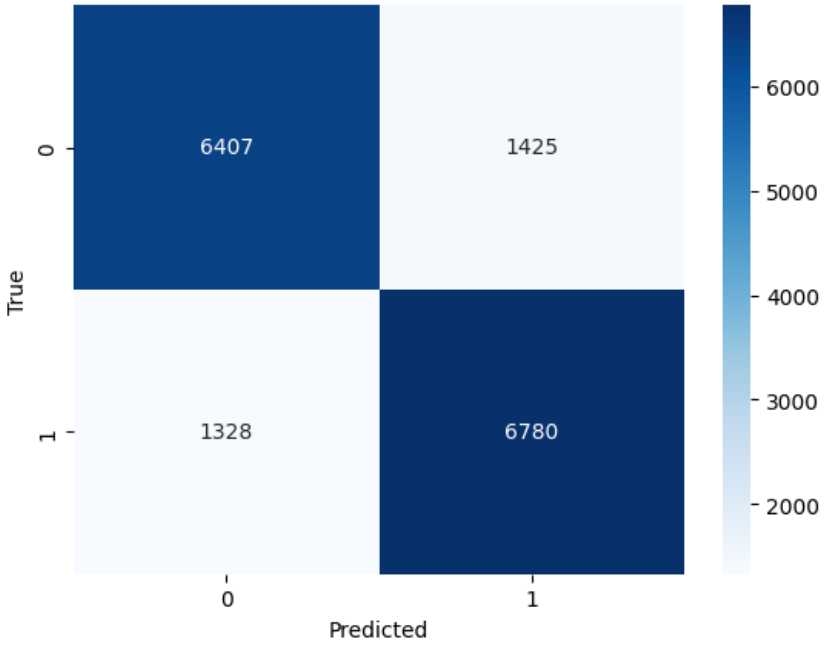
Al igual que antes, se prueba el modelo con parámetros por defecto. Para este modelo, se prueban con distintos "kernels".

Se crean los modelos con un kernel lineal, polinómico y radial, y se muestran las métricas. Las matrices de confusión resultantes son:



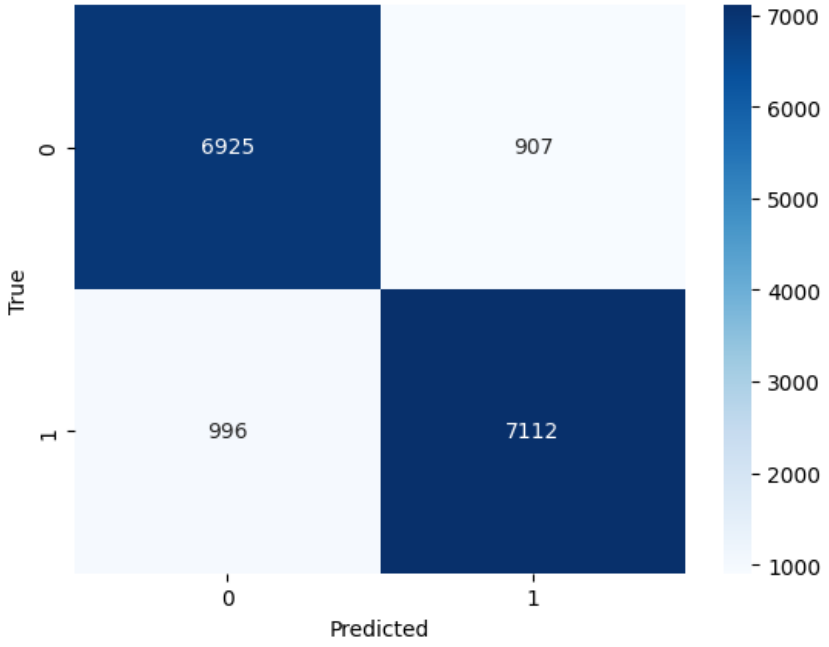
A continuación, se realiza una búsqueda de hiperparámetros con Random Search y utlizando un Bagging Classifier para disminuir el tiempo que tarda el modelo en entrenarse.

Se visualizan las métricas y se muestra la matriz de confusión:

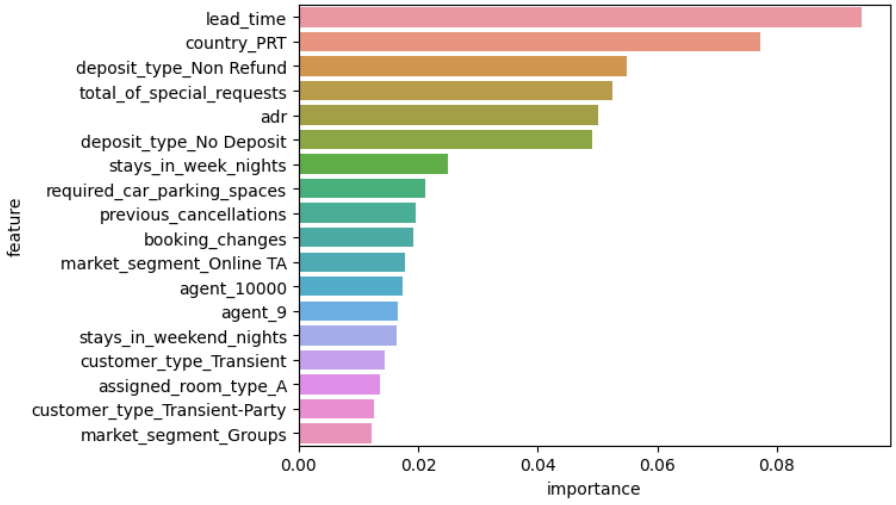


5 Random Forest (RF)

Al igual que antes, se crea el modelo, se buscan los hiperparámetros por Random Search Cross Validation, se muestran las métricas y se visualiza la matriz de confusión.

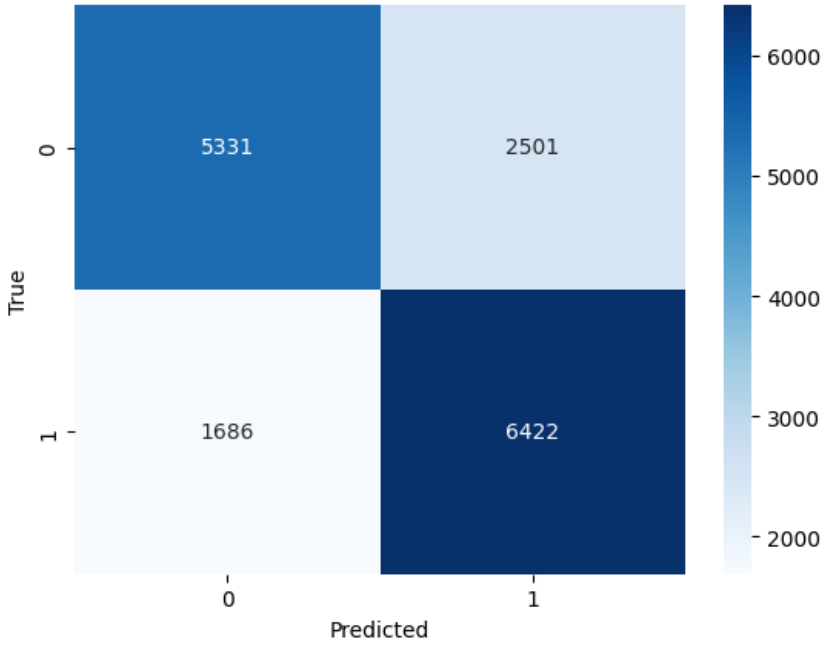


Como este modelo también utiliza árboles para tomar decisiones, y los parámetros se ordenan por orden de importancia, se puede mostrar el gráfico de la importancia de las features.



6 XGBoost

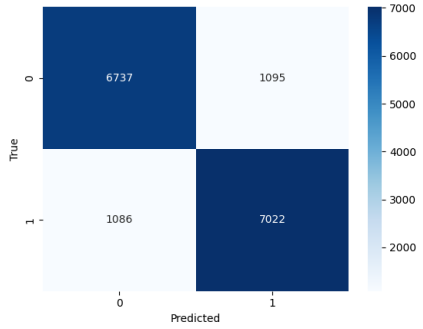
Al igual que antes, se crea el modelo, se buscan los hiperparámetros por Random Search Cross Validation, se muestran las métricas y se visualiza la matriz de confusión.

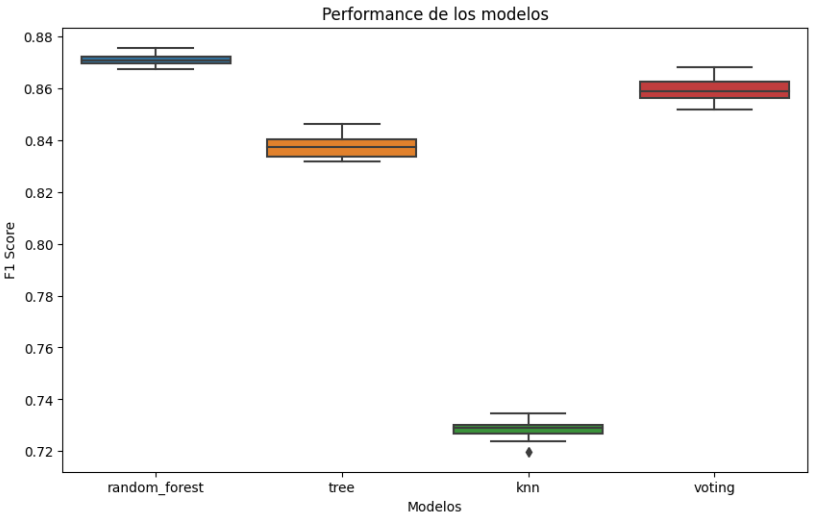


7 Voting Classifier

Se crea y entrena el voting classifier con un ensamble híbrido conformado por Random Forest, Decision Tree y KNN.

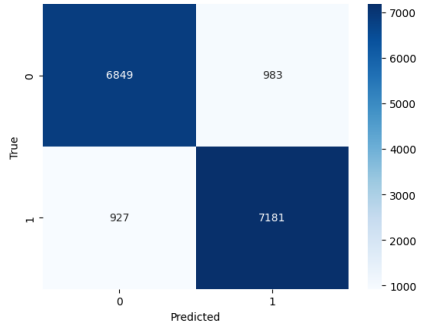
Se evalúan las métricas, y se visualiza la matriz de confusión y una comparación de la performance de los modelos individualmente y del voting classifier.

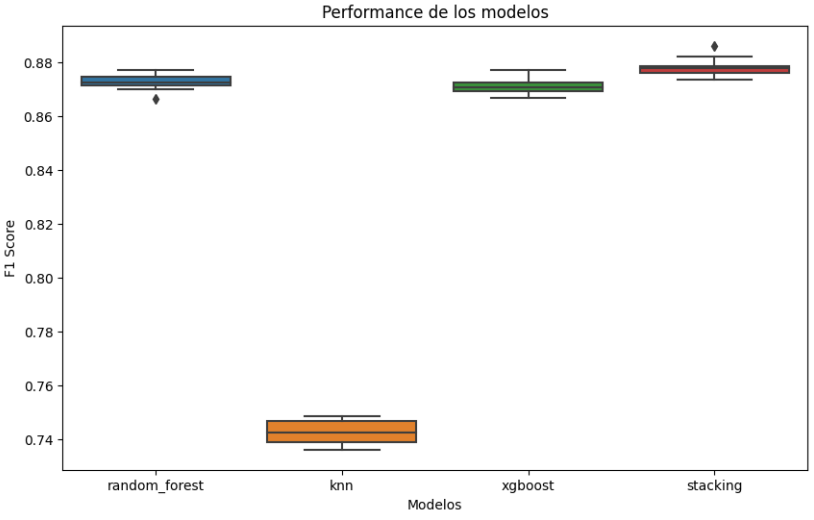




8 Stacking Classifier

Se realiza el mismo proceso para el stacking classifier, utilizando Random Forest, KNN y XGBoost, y se muestran los resultados:





9 Conclusiones

Esto da por concluida esta etapa del trabajo. Todos los modelos fueron debidamente guardados, junto a los resultados de las predicciones. Se detallan las métricas exactas en las correspondi-

entes secciones del notebook.

Con un score F1 del 88,3%, el Stacking Classifier es el modelo con mayor poder de predicción que elaboramos hasta ahora.