# Informe Final - Grupo 25

## Introducción

Durante el desarrollo de todo el Trabajo Práctico 1, desarrollamos modelos para lograr predecir la cancelación de una reserva de un hotel. Comenzando por el análisis exploratorio de los datos, visualización de los datos, tratamientos para datos faltantes y valores atípicos, para luego empezar a entrenar modelos de clasificación, efectuando para cada modelo la ingeniería de features necesaria.
Nuestro primer modelo fue sobre árboles de decisión y la mejora de sus hiper parámetros. Luego, continuamos con los clasificadores con hiper-parámetros mejorados: Regresión Logística, KNN, SVM, ensambles homogéneos Random Forest y XGBoost y los ensambles híbridos Stacking y Voting.
Como cierre del trabajo práctico, realizamos y optimizamos Redes Neuronales.
Se evaluó la performance de cada modelo y se subió su predicción a la competencia de Kaggle.

## Cuadro de Resultados

| Modelo | CHPN | F1-Test | Presicion Test | Recall Test | Kaggle |
|---|---|---|---|---|---|
| C4.5 (Árbol) | 2 | 0.78 | 0.82 | 0.73 | 0.79292 |
| Stacking | 3 | 0.84 | 0.86 | 0.82 | 0.8448 |
| Voting | 3 | 0.85 | 0.86 | 0.83 | 0.84435 |
| XGBoost | 3 | 0.85 | 0.86 | 0.84 | 0.8434 |
| Red Neuronal | 4 | 0.82 | 0.82 | 0.82 | 0.8238 |

**C4.5:** mejora del algoritmo ID3 para crear árboles de decisión, maneja valores faltantes e intervalos continuos.

**Stacking:** entrena varios modelos dado un conjunto de datos y luego entrena uno nuevo que, cuando reciba una nueva instancia de datos, determinará qué modelo es el más

adecuado para clasificar a dicha instancia (utilizamos KNN, RF y SVM con el meta_learner Regresión Logística).

**Voting:** entrena varios modelos distintos basados en el mismo dataset (en nuestro caso utilizamos XGBoost, RF y KNN). Todos los modelos realizan la predicción y la clasificación queda determinada por la salida más elegida.

**XGBoost:** está basado en boosting y construye un conjunto de árboles de forma secuencial, y cada uno se enfoca en corregir los errores de los anteriores. Incluye términos de regularización en la función de pérdida para controlar el sobreajuste y mejorar la generalización del modelo.

**Red Neuronal**: funcionan mediante la interconexión de neuronas artificiales, organizadas en capas. Cada conexión entre neuronas tiene un peso asociado que determina la influencia de una neurona sobre la otra, y estos pesos se ajustan para que la red aprenda patrones. La capa de entrada contiene el conjunto de entrada, las capas intermedias permiten que la red capture características relevantes y complejas en los datos y la capa de salida produce un resultado. En nuestro caso, el mejor modelo de red neuronal tiene dos capas con la misma cantidad de neuronas que las variables de entrada (64 neuronas y activación ReLu, regularización Dropout).

## Conclusiones generales

A partir del análisis exploratorio de los datos y la lectura del paper otorgado, analizamos todas las variables y eliminamos las que consideramos irrelevantes a nuestro objetivo (cancelación de la reserva). Graficamos ciertos atributos para observar su distribución y la correlación entre todos, esto permitió que viéramos claramente cuáles eran los más relacionados con el target. Analizamos los datos faltantes por filas y columnas y tomamos diferentes decisiones de tratamiento sobre ellos según el contexto, además, realizamos un estudio sobre los valores atípicos univariados y multivariados. Para avanzar en la construcción de modelos y predicciones, hicimos una recodificación de los datos. Estos tratamientos se aplicaron tanto en el archivo de entrenamiento como en el de test.

El análisis exploratorio y el preprocesamiento nos permitieron tener una visión general de los datos y prestar atención a ciertas variables más que otras. Además, nos permitió dejar el dataframe preparado para comenzar el entrenamiento de los modelos.

En el desarrollo posterior del trabajo, construimos modelos para lograr una predicción. Cada modelo fue hecho con un modelo base, y luego mejorado en la búsqueda de sus hiper-parámetros utilizando la mayoría de las veces Random Search, ya que nos permitía una búsqueda relativamente rápida y eficiente.

Hablando específicamente de cada modelo realizado, haremos un breve feedback sobre cada uno de ellos:

*Árboles de Decisión*: si bien obtuvimos un F1-Score de 0.77, encontramos bastante caro computacionalmente la mejora de hiper parámetros de este modelo, teniendo que dejar ejecutando la búsqueda por más de 8 hs.

*KNN*: este modelo es lo que consideramos moderado en cuanto a su rendimiento y costo computacional, mejoró bastante con la mejora de sus hiper parámetros.

*SVM*: realizamos un modelo base con datos no escalados, y luego escalamos y el rendimiento mejoró considerablemente. Luego realizamos 2 variantes, utilizando Kernel Lineal y Kernel Radial. Con el Kernel Lineal obtuvimos un resultado moderado, no muy variado al de KNN (incluyendo con sus mejores hiper-parámetros). Si hablamos del Kernel Radial, obtuvimos un buen F1-Score de 0.8, con la mejora de hiper-parámetros que resultó muy extensa en cuestión de tiempo de ejecución, sin embargo, no dio una buena predicción en Kaggle.

*Random Forest*: este modelo resultó muy efectivo sin necesidad de mejorar sus hiper-parámetros, ya que, una vez realizada la mejora de los mismos, no hubo mucha variación en la respuesta.

*XGBoost*: confirmamos lo estudiado teóricamente en la clase, este ensamble homogéneo se convirtió en el tercer modelo que mejor rindió (casi empatando con Voting), siendo rápido y a la vez eficiente.

*Stacking*: realizamos el ensamble con los clasificadores RF, SVM y KNN, y fuimos probando con varios meta-modelos y el que mejor rindió fue Regresión Logística (del cual también realizamos un análisis particular). Obtuvimos nuestro mejor score en Kaggle con este modelo, sin embargo, tuvo varias desventajas que nombraremos luego bajo nuestro punto de vista.

*Voting*: con este modelo, tuvimos problemas en su implementación al inicio, y no logramos una buena predicción. Además, no tuvimos en cuenta agregar nuestros

mejores modelos anteriores para lograr el mejor resultado. Una vez que encontramos los errores e intercambiamos los clasificadores por los mejores, obtuvimos una gran respuesta, eficiente y rápida (la segunda mejor).

*Redes Neuronales*: realizamos varios modelos de redes, pasando por modelos piramidales, modelos con misma cantidad de neuronas por capa, y variando la cantidad de capas. Como resultado obtuvimos una predicción eficiente, pero no fue la mejor. En base a lo estudiado, las redes neuronales requieren de grandes conjuntos de datos para entrenarse de forma correcta, además de que tienen mucha sensibilidad a hiper-parámetros que deben ajustarse adecuadamente y esto puede ser una tarea complicada y costosa que podemos dejársela a cargo a los expertos en la materia. También, las redes neuronales pueden ser propensas al sobreajuste con datos pequeños, esto lo hemos visto en la práctica ya que en varios de nuestros modelos a pesar de usar regularizadores, con nuestro conjunto de entrenamiento obtuvimos un score que luego en Kaggle bajaba. Incluso, hicimos uso de random search para mejorar nuestro score en kaggle, buscando una mejor cantidad de 'epochs' y 'batch_size', ampliando la cantidad de particiones para el cross validation pero aún así, ésta no aumentó.

El modelo que **mejor rindió en Kaggle** fue Stacking, pero más allá de eso, debemos decir que su entrenamiento no fue el más sencillo de todos, tardó bastante tiempo. Por eso decidimos quedarnos con XGBoost como el mejor modelo de clasificación, siendo este más sencillo de entrenar y no perder eficacia por eso, además también fue el que **mejor rindió en Test**.

Ahora si nos centramos en la **rapidez de entrenamiento**, podríamos decir que Voting es una excelente respuesta, sí no tomamos en cuenta el tiempo que nos llevó optimizar los modelos que ensamblamos. Tomó un tiempo de menos de 2 minutos de entrenamiento para poder observar las métricas y resultó ser (una vez ensamblados los modelos "correctos") un eficaz predictor. Depende bastante de qué modelos uno ensambla.

Si queremos un modelo rápido que no sea un ensamble, podríamos usar Regresión Logística, al menos de la forma en el cual lo construimos, tardó muy poco tiempo y estaba optimizado. Eso no quiere decir que sea el mejor predictor, pero tiene a su favor el poco tiempo de construcción.

Si tomamos en cuenta que logramos el mejor score en Kaggle con el valor de 0.84 aproximadamente, en cuestión de métricas, los modelos logrados han tenido un

buen rendimiento. Pensamos que en el ámbito productivo podríamos encontrarnos con datasets tal vez mucho más extensos, lo que nos hace pensar si no se vería afectada la escalabilidad de nuestros modelos, los cuales deberían ser capaces de manejar datos a gran escala sin una degradación significativa de su desempeño. Además, nos topamos varias veces con límites computacionales a la hora de entrenar modelos, siendo esta una razón para pensar en que los recursos de hardware insuficientes pueden afectar la capacidad del modelo para procesar datos y realizar predicciones.

Pensamos que siempre podemos mejorar los resultados, una forma sería seguir evaluando y refinando el modelo a medida que obtengamos más datos, siendo estos procesados de la manera adecuada y tomando en cuenta el feedback obtenido, para lograr obtener la mayor interpretación de los datos posible. Tal vez como en varias ocasiones tuvimos problemas con el sobreajuste, priorizar las técnicas de regularización y tomar más en cuenta el desequilibrio de las clases, mejorarían la eficacia de predicción.

## Tareas Realizadas

Teniendo en cuenta que el trabajo práctico tuvo una duración de 9 (nueve) semanas, le pedimos a cada integrante que indique cuántas horas (en promedio) considera que dedicó semanalmente al TP.

| Integrante | Promedio Semanal (hs) |
|---|---|
| Mariana Juarez Goldemberg | 14 horas |
| Miranda Marenzi | 15 horas |
| Lisandro Roman | 12 horas |