Sprint 4: Caso de uso

Predicción de atrasos en vuelos

Abril, 2023

Integrantes:

BERENGUEL, Rafael

CATANIA, Ezequiel

FIORINO, Santiago

GARCÍA, Nicolás Francisco

IGLESIAS, Juan Martín

LINIADO, Dylan

ORTEGANO, Rafael

PALACIOS, Martín

PALMIERI, Marco

ROSALES, Gustavo

Índice

1. Entendimiento del problema

La puntualidad de los vuelos es un factor crítico para la satisfacción de los pasajeros y la eficiencia en la industria aérea. Sin embargo, es común que los vuelos experimenten atrasos, lo que puede tener consecuencias negativas tanto para los pasajeros como para las aerolíneas. Los atrasos pueden generar costos adicionales para las aerolíneas, así como la necesidad de reprogramar vuelos y pagar compensaciones a los pasajeros afectados. Asimismo, ellos pueden experimentar estrés y pérdida de tiempo debido a los atrasos, lo que puede afectar su experiencia de viaje y su fidelidad a las aerolíneas.

En este contexto, vamos a desarrollar distintos modelos de *machine learning* con el objetivo de predecir atrasos en los vuelos y, por lo tanto, permitir que las aerolíneas tomen medidas proactivas para evitarlos o mitigar sus consecuencias. Estos modelos usarán datos históricos de los vuelos, en conjunto con factores externos como el clima y las distancias.

Para nuestro desarrollo, se consideran a los vuelos como **atrasados** cuando la diferencia entre la hora para la cual el vuelo estaba programado y la hora en la que realmente salió es mayor a 15 minutos. Este criterio ha sido fijado por parte del cliente. Las consecuencias de que un vuelo se retrase menos de 15 minutos no tienen un gran impacto, mientras que si un vuelo se retrasa más de 15 minutos, ya empieza a producir los problemas mencionados anteriormente.

2. Análisis de datos

Los datos de los vuelos fueron provistos por el cliente, en un *dataset* de casi 70 mil registros de vuelos. Estos registros incluyen la hora de salida programada, la hora de salida real, el destino, la aerolínea, entre otros. Todos los vuelos son del año 2017, y su origen es el Aeropuerto Internacional de Santiago (SCEL).

Para cada vuelo se cuenta con la siguiente información:

- **Fecha-I:** Fecha y hora programada del vuelo.
- **Vlo-I:** Número de vuelo programado.
- **Ori-I:** Código de ciudad de origen programado.
- **Des-I:** Código de ciudad de destino programado.
- **Emp-I:** Código aerolínea de vuelo programado.
- **Fecha-O:** Fecha y hora de operación del vuelo.
- **Vlo-O:** Número de operación del vuelo.
- **Ori-O:** Código de ciudad de origen de operación.
- **Des-O:** Código de ciudad de destino de operación.
- **Emp-O:** Código aerolínea de vuelo operado.
- **DIA:** Día del mes de operación del vuelo.
- **MES:** Número de mes de operación del vuelo.
- **AÑO:** Año de operación del vuelo.
- **DIANOM:** Día de la semana de operación del vuelo.
- **TIPOVUELO:** Tipo de vuelo, I= Internacional, N= Nacional
- **OPERA:** Nombre de aerolínea de origen.
- **SIGLAORI:** Nombre ciudad origen.
- **SIGLADES:** Nombre ciudad destino.

2.1. Análisis general

En nuestro análisis de datos, lo primero que buscamos fue conocer la distribución de los vuelos atrasados frente a los que no lo están. Para esto, revisamos nuestra variable objetivo y encontramos que el 18.5 % de los vuelos están atrasados más de 15 minutos, mientras que el 81.5 % no lo están. A continuación, mostramos un gráfico tipo torta que ilustra esta distribución.

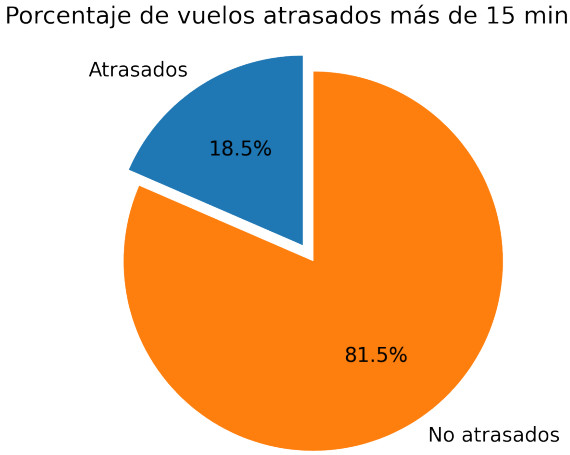


Figura 1: Porcentaje de vuelos atrasados

Como podemos visualizar, la mayoría de los vuelos no presentan atrasos significativos. Sin embargo, nuestro interés se focaliza en la identificación de aquellos vuelos que sí presentan retrasos, ya que ello puede tener un impacto considerable en la experiencia del pasajero y en la planificación de vuelos de la aerolínea. Con este objetivo en mente, buscaremos crear modelos de predicción que nos permitan identificar estos casos con la mayor precisión posible.

En nuestro análisis exploratorio de datos, hemos observado que es importante tener en cuenta la distribución temporal de los vuelos. Por ello, hemos evaluado tres variables: mes, día de la semana y período del día, para entender cómo se distribuyen los vuelos en nuestro dataset. El objetivo de este análisis es ver si existen meses, días de la semana u horas del día en las que se concentran la mayoría de los vuelos, o por el contrario, si hay períodos con poca cantidad de datos.

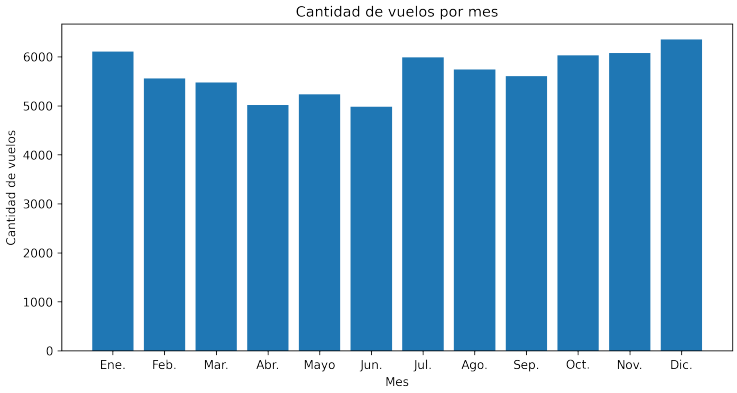


Figura 2: Cantidad de vuelos por mes

En cuanto a los meses, podemos ver que la cantidad de vuelos por mes en nuestro dataset está bastante equilibrada.Sin embargo, en Enero, Julio y Diciembre encontramos un pequeño aumento en la cantidad de vuelos registrados, lo cual tiene sentido considerando que son meses en los que se suele tomar vacaciones o conforman la "temporada alta."[en] el turismo. En general, los datos son suficientes para realizar un análisis confiable.

Como se mencionó anteriormente, se buscó identificar los días de la semana con mayor y menor cantidad de vuelos programados en nuestro dataset. Para ello, se generó un gráfico de barras que muestra la proporción de vuelos por día de la semana.

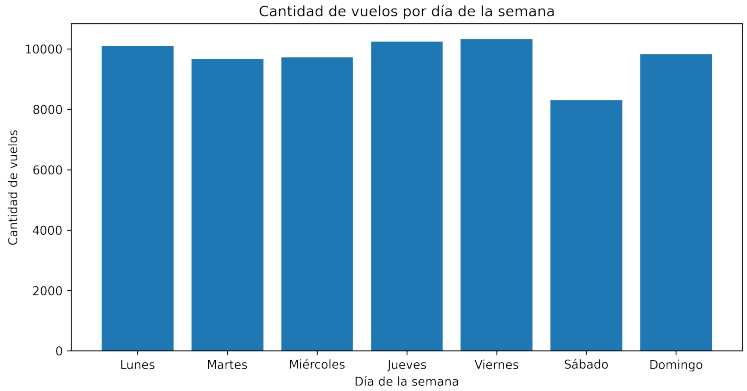


Figura 3: Cantidad de vuelos por día de la semana

Como se puede observar en el gráfico, los días Jueves y Viernes tienen la mayor proporción de vuelos programados en comparación con los otros días de la semana. En contraste, el Lunes es el día con menor cantidad de vuelos. Estos hallazgos podrían ser útiles para las aerolíneas al momento de planificar sus operaciones y optimizar sus recursos, considerando que nuestro objetivo está en poder predecir la previsión de atrasos de vuelos. Estos datos son útiles debido a que nos permiten ver que, si bien la diferencia no es tan grande, hay que tener en cuenta el día de la semana para poder entender mejor el contexto de los datos.

En el siguiente gráfico se identifica la distribución de los vuelos a lo largo del día. Para esto, se realizó un gráfico de barras que indica si el vuelo es en la mañana, tarde o noche.

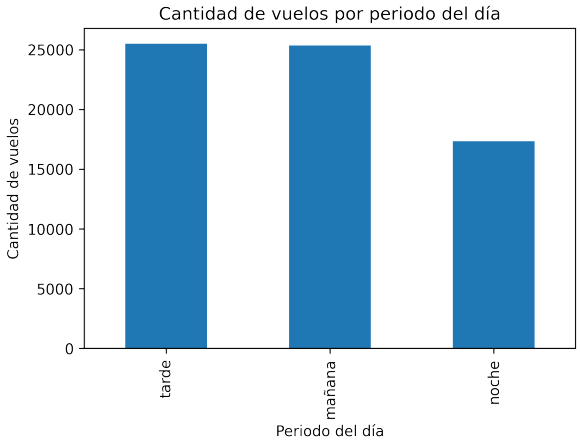


Figura 4: Cantidad de vuelos por periodo del día

A partir de este análisis, se puede concluir que la mayoría de los vuelos programados en el conjunto de datos se realiza en el período de la tarde, seguido de cerca por los vuelos de la mañana. Por otro lado, el período de la noche concentra una menor cantidad de vuelos. Esta cifra se alcanza ya que durante el día es cuando hay más movimiento de personas en el aeropuerto.

Luego también hicimos un análisis de los datos geográficos con los que contamos, como el destino y tipo de vuelo.

Del total de vuelos, el 54.2 % son nacionales y el 45.8 % son internacionales.

Contamos con 62 destinos, de los cuales 16 son nacionales y 46 son internacionales. 20 de los 62 destinos cuentan con más de 1000 vuelos durante el año.

2.2. Composición de la tasa de atrasos

La tasa de atrasos, además de definirse genéricamente como la relación entre la cantidad y la frecuencia de un evento o conjunto de fenómenos previamente definidos (estadísticamente), éste indica el porcentaje de vuelos que salen con un retraso (en nuestro caso, se definió vuelo atrasado a aquel que tardó 15 minutos en despegar) respecto a su hora programada de salida. Es llamativo agregar que no todos los retrasos son responsabilidad de las aerolíneas. Algunos pueden ser causados por factores externos, como el clima, la distancia o la congestión del tráfico aéreo. Sin embargo, las aerolíneas tienen la responsabilidad de minimizar ese porcentaje. A fin y al cabo, tal objetivo se determina dentro de la lógica de negocio.

Conocer cómo varía la tasa de atrasos en función de distintas variables, como la temporada o el tipo de vuelo, nos permite entender mejor el comportamiento de los datos y seleccionar las variables más importantes para nuestro modelo de predicción de atrasos. Al tener en cuenta estas variables en nuestro modelo, podremos crear una herramienta de predicción más precisa. Por eso dedicamos esta sección a estudiar la tasa de atrasos en función de distintas variables.

Respecto a la tasa de atraso según el destino, la mayoría de los destinos presentan una tasa de atraso entre 0.1 y 0.4 . Pudimos notar 8 excepciones, 4 destinos con 100 % de tasa de atraso y otros 4 destinos con 0 %. Analizando más a profundidad, notamos que dichos destinos solo contaban con 1 vuelo, lo cual explica esos valores ya que si ese único vuelo se atrasó, queda 100 % y si no se retrasó da 0 %.

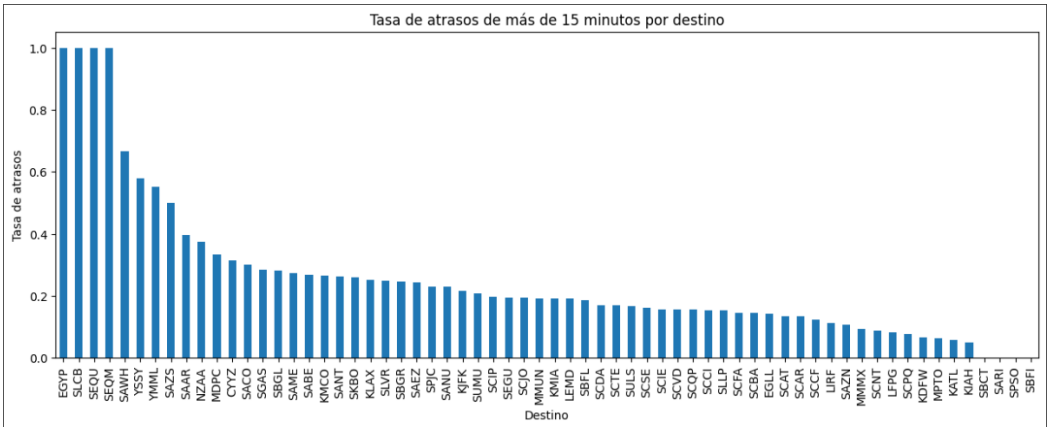


Figura 5: Tasa de atrasos de vuelos por destino

Los 4 destinos con mayor tasa de atraso son Ushuaia, Sydney, Melbourne y Bariloche con 0.66, 0.57, 0.55 y 0.5 respectivamente. A su vez, los 4 destinos con menor tasa de atraso son Dallas (0.06), Panamá (0.06), Atlanta (0.05) y Houston(0.05).

Luego analizamos cómo es la tasa de atrasos en temporada alta y baja. En la temporada alta, la tasa de atrasos fue del 19.64 %, mientras que en la temporada baja fue del 17.91 %. Como era de esperar, en la temporada alta es más alta, ya que hay una mayor cantidad de pasajeros y vuelos, lo que lleva a una mayor congestión y posibles demoras.

Posteriormente,visualizaremos la tasa de atrasos por empresa/aerolínea, la cual nos indica aquellas que presentaron mayor cantidad de vuelos retrasados. Este parámetro impacta significativamente en la reputación de la compañía en particular,permitiendo que en un corto lapso de tiempo sus ingresos se reduzcan notablemente. De acuerdo a lo obtenido en el gráfico, GRUPO LATAM es la que presenta mayor porcentaje de atrasos (100 %), mientras que AeroMexico es la que menos inconvenientes presenta, estando por debajo del 0.2 % de atrasos.

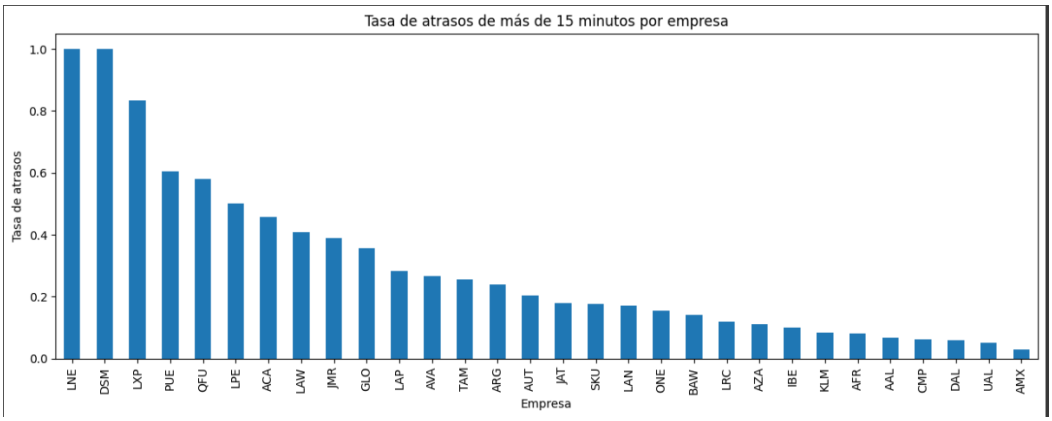


Figura 6: Tasa de atrasos de vuelos por aerolínea

Ahora bien, ¿qué sucede con la distribución de los atrasos según diferentes çategorizaciones"de tiempo? Para ello, partiremos de mayor a menor. Con respecto a los meses, la mayor tasa de atrasos se generó en el mes de Julio, alcanzando un 30 %, mientras que Marzo alcanzó un 12 %. Esta métrica, a lo largo de todo el desarrollo del proyecto, podrá ser utilizada como feature para poder entrenar nuestros modelos de inteligencia artificial. Con respecto a las horas transcurridas durante el día, los vuelos presentan atrasos mayormente a las 21:00hs y las 00:00hs. Sin embargo, teniendo en cuenta la cantidad de vuelos que se realizaron en diferentes períodos del día (mañana, tarde, noche), se registró mayor cantidad de vuelos desde la mañana hasta la tarde. Por lo tanto, no se cumple cierta proporcionalidad que indique que a mayor cantidad de vuelos durante un período del día, mayor tasa de atrasos.

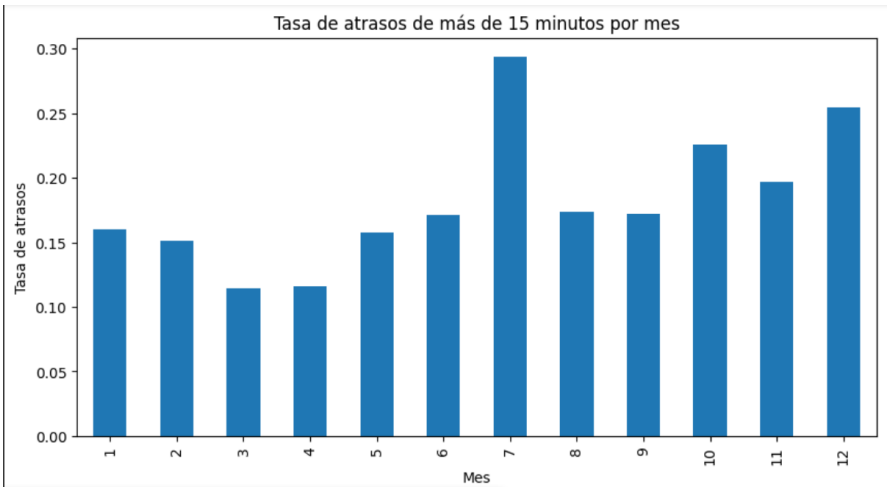


Figura 7: Tasa de atrasos de vuelos por mes

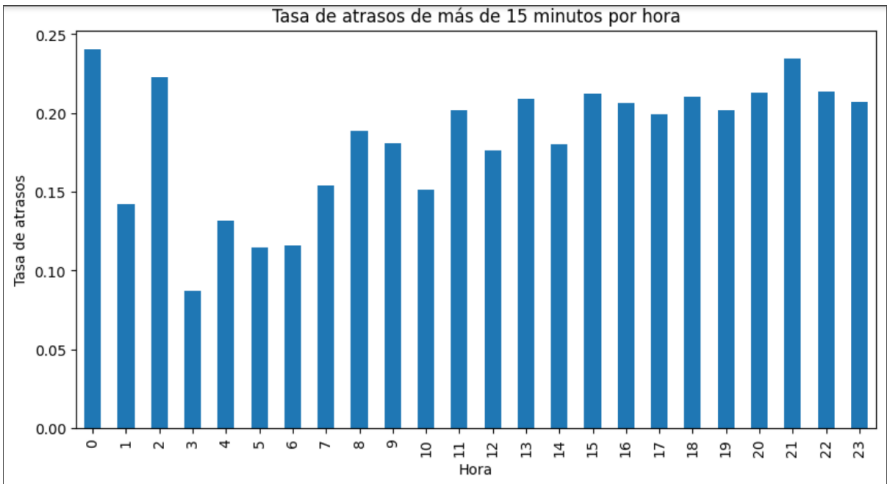


Figura 8: Tasa de atrasos de vuelos por hora

Para finalizar, analizamos como varía la tasa de atrasos en función de si un vuelo es internacional o nacional. Obtuvimos que en los vuelos internacionales, la tasa de atrasos es del 22.56 %, mientras que la tasa de atrasos en vuelos nacionales es del 15.04 %. Esto era esperable, ya que los vuelos internacionales son de mayor duración, requiriendo más condiciones previas al despegue.

3. Extensión de datos

En los modelos de *machine learning*, para lograr un alto rendimiento, es importante contar con variables que representen adecuadamente las características de los datos y capturen la mayor cantidad posible de información relevante.

En el contexto de la predicción de atrasos en vuelos, existen diversas variables que pueden ser relevantes para el modelo. Por ejemplo, el horario de salida, la aerolínea y el tipo de vuelo pueden influir en la probabilidad de un atraso. Sin embargo, es posible que estas variables por sí solas no sean suficientes para capturar la complejidad del problema y obtener un alto rendimiento del modelo. Es por esto que en esta sección presentaremos una serie de variables agregadas. Estas variables pueden proporcionar información adicional y ayudar al modelo a capturar relaciones más complejas entre las características de los vuelos y los atrasos.

Las variables agregadas que presentamos incluyen características como la distancia a los destinos, el clima y la cantidad de atrasos en los vuelos inmediatamente anteriores. A través de la inclusión de estas variables agregadas, esperamos mejorar significativamente el rendimiento de nuestro modelo y proporcionar predicciones más precisas sobre la probabilidad de un atraso en un vuelo.

3.1. Variables Agregadas

El cliente nos brindó una lista de variables a agregar que, en base a su entendimiento del dominio del problema, decidieron que serán útiles para alcanzar un buen rendimiento. Esas variables son las siguientes:

- **Temporada alta:** Variable binaria que indica si el vuelo está en temporada alta. Hay tres rangos de fechas considerados temporada alta, los cuales son:
  1. Entre el 15 de Diciembre y el 3 de Marzo
  2. Entre el 15 y el 30 de Julio
  3. Entre el 11 y el 30 de Septiembre
- **Diferencia en minutos:** La diferencia de minutos entre la hora de salida programada, y la hora de salida real.
- **15 minutos atrasados:** Variable binaria que indica si la diferencia en minutos es mayor a 15. Esta será la variable a predecir.
- **Periodo del día:** Una variable categórica en base a la hora, que puede tomar tres valores:
  1. Mañana: Entre las 05:00 y las 11:59
  2. Tarde: Entre las 12:00 y las 18:59
  3. Noche: Entre las 19:00 y las 04:59

Además, a partir del entendimiento de los datos, hemos decidido desarrollar nuestras propias variables para enriquecer nuestro set de datos con información más detallada sobre los atrasos.

- **Atrasos en últimos 15 vuelos:** Es la sumatoria de vuelos con retrasos que hubieron en los últimos 15 vuelos anteriores a cada periodo del dataset.
- **Minutos desde el último atraso:** Es la cantidad de minutos transcurridos desde el último retraso ocurrido.
- **Promedio de minutos entre atrasos anteriores:** Es el promedio móvil de minutos transcurridos entre los últimos 10 vuelos atrasados anteriores.

3.2. Datos Externos

3.2.1. Distancia

La distancia entre los aeropuertos de origen y destino es un factor clave que puede influir en la puntualidad de los vuelos. Los vuelos más largos tienden a tener mayor retraso, como se ha visto anteriormente que la tasa de atrasos en vuelos internacionales es superior a la de vuelos locales, es decir, de menor distancia. Esto puede deberse a por ejemplo, factores como el clima, el tráfico aéreo y las interrupciones en la operación de la aerolínea.

Por esa misma razón, agregamos a nuestro *dataset* las distancias a cada uno de los aeropuertos. La página web *Flight Plan Database* [1] brinda un servicio en el cuál le podes indicar dos códigos de aeropuertos (estándar), y te devuelve la distancia. La distancia está claramente correlacionada con la duración del vuelo, y esperamos que esta información mejore la precisión de las predicciones de atrasos en vuelos.

[1] *flightplandatabase.com*

3.2.2. Clima

El clima es uno de los factores más importantes a considerar en la predicción de atrasos, ya que puede afectar significativamente las operaciones de las aerolíneas. El mal clima puede causar demoras, desvíos, cancelaciones y otros problemas que pueden influir en la puntualidad de los vuelos.

Si bien el clima influye tanto en el origen como en el destino, para poder incluir los datos del clima del destino, tendríamos que tener acceso a los datos climáticos de cientos de aeropuertos, ya que los registros incluyen múltiples destinos. Por esa misma razón, decidimos tener en cuenta únicamente los datos climáticos del origen. Accedimos a la información de la Dirección Meteorológica de Chile[2] para agregar a nuestro *dataset* los datos del clima del Aeropuerto Internacional de Santiago. Los datos incluyen:

- **Humedad**
- **Presión**
- **Temperatura**
- **Viento:** Dirección y fuerza
- **RRR6:** Cantidad de precipitación líquida caída o acumulada durante un período de 6 horas.
- **Traza:** Cantidad de lluvia, nieve u otro tipo de precipitación mayor que cero, pero demasiado pequeña para ser medida por un instrumento de medición estándar como un pluviómetro.

En el caso de las variables **RRR6** y **Traza**, las mediciones fueron hechas cada 6 horas, pero los vuelos pueden ser a cualquier hora. Para solucionar esto decidimos rellenar los datos faltantes usando una *interpolación lineal*, ya que si a las 00:00 se midió 0 de precipitación, y a las 06:00 se midió 10, en algún momento tuvo que pasar por los valores intermedios.

4. Preparación de datos

Durante el proceso de preparación y limpieza del dataset, se realizaron varias correcciones para asegurar la calidad de los datos. En particular, se realizó una corrección de valores nulos en el conjunto de datos, lo que implicó la eliminación de filas con información faltante. También se llevó a cabo la eliminación de filas en las que el número de vuelo programado y operado no coincidían, ya que esto podría generar errores en el análisis posterior.

Además, se realizaron otras modificaciones en el dataset para asegurar la integridad de los datos, lo que incluyó la eliminación de valores atípicos y la estandarización de los nombres de las variables. Estas correcciones fueron necesarias para garantizar que los modelos de Machine Learning que se entrenaran estuvieran basados en datos precisos y confiables, lo que permitió obtener resultados más precisos y fiables en la predicción de retrasos en los vuelos.

4.1. Elección de variables a utilizar

La elección de variables es un paso crucial en cualquier análisis estadístico o modelo predictivo. Se refiere al proceso de seleccionar las variables más relevantes y significativas que se utilizarán en el modelo, lo que puede tener un gran impacto en la precisión y eficacia del resultado final. La elección cuidadosa de variables puede ayudar a evitar el sobreajuste o la subestimación, y también puede reducir el desarrollo y los tiempos necesarios para realizar los análisis de los modelos.

En el proceso del modelado, es común que se realice una selección cuidadosa de variables para garantizar la precisión y eficacia del resultado final. Sin embargo, en ocasiones puede suceder que, después de probar diversas combinaciones de variables, se llegue a la conclusión de que la inclusión de todas las variables utilizadas en el modelo proporciona el mejor resultado posible. Es decir, que cada una de las variables seleccionadas tiene un impacto positivo y significativo en el resultado final.

[2] *climatologia.meteochile.gob.cl*

En este sentido, es importante destacar que la inclusión de todas las variables no significa necesariamente que se haya realizado un análisis superficial o poco riguroso. Al contrario, puede ser una muestra de un proceso de selección cuidadoso y exhaustivo, donde cada variable ha sido considerada y evaluada de manera individual y en conjunto con las demás. Con el paso de las pruebas se fueron agregando variables nuevas que llevaron al modelo a seguir siendo aún mas efectivo.

Finalmente, para la creación del modelo seleccionamos la mayoría de las features originales, las solicitadas y las agregadas por nosotros: Vlo I, Fecha I, Des I, dianom, Emp I, MES, Des O, Emp O, opera, tipovuelo, periodo dia, temporada alta, distancia, hora, atraso 15, humedad, presion, temperatura, viento ff, viento dd, RRR6 interpolated, Traza interpolated, sum atraso 15, promedio min atraso anteriores y dif min atraso.

4.2. Pasaje de categorías a números

Durante el desarrollo de los modelos se probaron diferentes métodos para transformar nuestras variables categóricas en variables numéricas. Esto es necesario ya que los modelos de aprendizaje automático solo pueden trabajar con números, no con cadenas de texto, fechas, ni otros tipos de datos.

Existen distintos métodos para realizar esa transformación. En una primera instancia se probó el método *One Hot Encoding* [3], que crea una nueva variable para cada posible valor de la variable categórica y establece el valor de la variable en 1 si la observación pertenece a esa categoría, o en 0 si no. Sin embargo, dada la gran cantidad de clases que se tenían, se creaban demasiadas columnas, las cuales por sí solas no aportan gran información. Una posible solución para reducir la cantidad de columnas, es usando técnicas como *PCA* [4] para reducir la dimensión sin perder información valiosa. Pero incluso después de reducir el número de features con técnicas como PCA, los resultados siguieron sin mejorar.

Para lidiar con este problema se usó el método *Label Encoding* [5], el cual convierte cada columna categórica a una numérica asignando un número a cada clase. Este método, al convertir las columnas categóricas en columnas numéricas, no incrementa el número de features y nos dió mejores resultados que *One Hot Encoding.*

Finalmente, la técnica por la que nos decantamos fue el *Mean Encoding*, utilizando una *media móvil exponencial con una ventana de tiempo.* El resultado fue significativamente superior a los otros métodos. Esta técnica consiste en, por cada vuelo, fijarnos por cada clase a la que queremos reemplazar por un valor numérico, el promedio móvil exponencial de vuelos con esa clase que se retrasó en el pasado. Solo tenemos en cuenta vuelos pasados en un marco temporal de 150 periodos. Esta fue la transformación que nos dio mejores resultados.

4.3. Normalizado

Normalizar los datos es un paso importante en el proceso de entrenamiento de modelos de aprendizaje automático. El objetivo de la normalización es ajustar los datos a una escala común para que todas las variables contribuyan igualmente al modelo, independientemente de su escala original. Sin la normalización, las variables con escalas mayores dominarán las variables con escalas menores. Esto puede afectar negativamente el rendimiento del modelo.

Existen varias técnicas de normalización, como la normalización z-score, la normalización por máximos y mínimos (MinMax), la normalización por escala media y la normalización por rangos. Probamos estas técnicas en nuestro proyecto, pero encontramos que no afectaban significativamente el rendimiento de los modelos. Por lo tanto, decidimos normalizar los datos con la técnica *MinMax* [6] ya que, si bien no cambia mucho respecto a las demás, fue la que nos brindó el mejor rendimiento. Esta técnica escala los datos a un rango de 0 a 1.

[3] *https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html*
[4] *https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html*
[5] *https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html*
[6] *https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html*

Un problema con el que nos cruzamos en cuanto a la normalización es que inicialmente la hacíamos previamente a la separación de los datos de entrenamiento y testeo. Luego nos dimos cuenta que es importante realizar la normalización después, para evitar *leaks* (fuga de información). Si se normaliza antes de separar, se estarían usando datos del conjunto de testeo para calcular estadísticas necesarias para el normalizado. Por lo tanto, normalizamos los datos después de la separación para garantizar que el proceso sea evaluado de manera justa.

4.4. Balanceo

El balanceo de datos es una técnica utilizada en el aprendizaje automático para abordar conjuntos de datos desequilibrados, es decir, cuando una de las clases tiene muchas más observaciones que la otra. En nuestro caso, como se ve en la Figura 1, solo el 18.5 % de nuestros datos son de vuelos atrasados, lo que puede llevar a un sesgo en nuestro modelo y a una mala predicción de los vuelos atrasados.

Las técnicas de balanceo se separan en dos grupos: *Undersampling* y *Oversampling*. En el caso de *undersampling*, se balancea el *dataset* mediante la reducción de la clase mayoritaria, es decir, borrando datos de la clase que domina. En cambio, el *oversampling* implica la replicación o generación sintética de observaciones de la clase minoritaria.

Para este caso de uso, utilizamos tanto técnicas de *undersampling* como de *oversampling*, que serán mencionadas a continuación. Todos los modelos que desarrollamos fueron probados con cada una de estas técnicas.

4.4.1. Undersampling

- **Random Undersampling:** RUS [6] selecciona al azar un subconjunto de muestras de la clase mayoritaria para igualar el número de muestras en ambas clases. Aunque es una técnica simple y rápida, tiene el riesgo de perder información importante y patrones en los datos, ya que se eliminan aleatoriamente muestras de la clase mayoritaria.
- **Edited Nearest Neighbours:** ENN [2] elimina muestras de la clase mayoritaria que son clasificadas incorrectamente por sus vecinos más cercanos en la misma clase. Es decir, se eliminan muestras que son ruido o atípicas.
- **Neighbourhood Cleaning Rule:** NCR [4] es similar a la técnica anterior, pero en lugar de eliminar directamente las muestras, primero se identifican los vecinos más cercanos en ambas clases y se eliminan los que están mal clasificados. También se eliminan muestras de la clase minoritaria que tienen vecinos de la clase mayoritaria, ya que se consideran ruido.

4.4.2. Oversampling

- **Synthetic Minority Oversampling Technique:** SMOTE [8] crea muestras sintéticas de la clase minoritaria, interpolando instancias similares. Primero selecciona una instancia de la clase minoritaria y calcula los vecinos más cercanos en el conjunto de entrenamiento. Luego elige uno de los vecinos cercanos y genera una nueva instancia (sintética, interpolada) en el espacio entre la instancia seleccionada y el vecino elegido.
- **SMOTE + RUS:** Por último, probamos combinando *undersampling* y *oversampling*, en este caso, las técnicas SMOTE y RUS.

5. Modelos

Para el modelado construimos y evaluamos diversos modelos de *machine learning* como Regresión Logística, Decision Tree, Random Forest, XGBoost, CatBoost y redes neuronales.

El dataset fue dividido en un conjunto de train (80 %) y en un conjunto de test (20 %).

5.1. Métricas

Antes de mostrar los modelos, es importante entender las métricas que se utilizarán para evaluar su rendimiento. En el proceso de entrenamiento de un modelo de *machine learning*, es crucial contar con una forma de medir su capacidad para hacer predicciones precisas. Las métricas permiten comparar diferentes modelos y ajustar parámetros para lograr el mejor resultado posible. En esta sección, explicaremos las métricas de evaluación de modelos que se utilizarán en nuestro proyecto y cómo interpretar sus valores para determinar el rendimiento del modelo.

Con las métricas que presentaremos a continuación, evaluamos y comparamos cómo funciona cada modelo para el conjunto de entrenamiento y el de testeo, verificando que no haya *overfitting*.

5.1.1. Matriz de confusión

La matriz de confusión es una tabla que se utiliza en problemas de clasificación para evaluar dónde se cometieron errores en el modelo. Las filas representan las clases reales que deberían haber sido los resultados, mientras que las columnas representan las predicciones que hemos hecho. Haciendo uso de esta matriz, resulta sencillo visualizar qué predicciones están equivocadas.

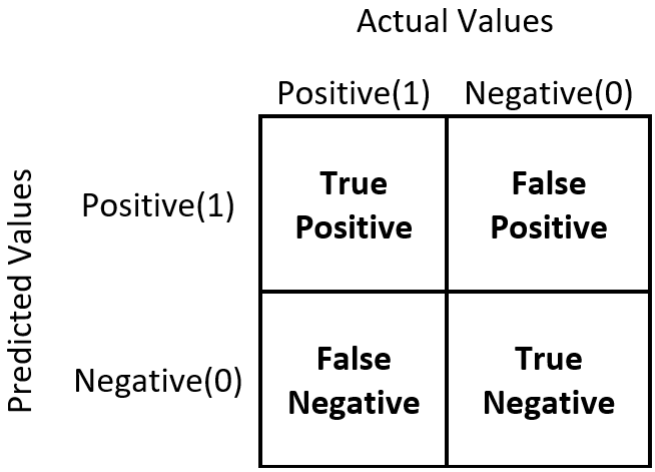


Figura 9: Matriz de confusión

Esta matriz es la base sobre la que se construyen todas las métricas de clasificación.

- **Accuracy:** La métrica accuracy representa el porcentaje total de valores correctamente clasificados, tanto positivos como negativos.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

- **Precision:** La métrica de precisión representa el porcentaje de valores que se han clasificado como positivos que son realmente positivos.

$$Precision = \frac{TP}{TP+FP}$$

- **Recall:** La métrica de recall, también conocida como el ratio de verdaderos positivos, es utilizada para saber cuántos valores positivos son correctamente clasificados.

$$Recall = \frac{TP}{TP+FN}$$

- **F1 Score:** Esta métrica combina el precision y el recall, para obtener un valor mucho más objetivo.

$$F1Score = 2.\frac{recall.precision}{recall+precision}$$

5.1.2. Curva ROC y AUC

La curva ROC (Receiver Operating Characteristic) es un gráfico muy utilizado para evaluar modelos de *machine learning* para problemas de clasificación. La gráfica representa la tasa de verdaderos positivos (True Positive Rate) en función de la tasa de falsos positivos (False Positive Rate).

El área bajo la curva ROC (AUC) es una medida de la capacidad del modelo para distinguir entre clases positivas y negativas. La AUC mide el área debajo de la curva ROC y varía entre 0 y 1, donde un valor de 0.5 indica que el modelo no es mejor que un clasificador aleatorio, y un valor de 1 indica un modelo perfecto que puede distinguir entre clases positivas y negativas con una precisión del 100 %.

5.2. Decision Tree

El árbol de decisión (*Decision Tree*) [9] es un modelo de aprendizaje automático supervisado que permite predecir el valor de una variable objetivo mediante la construcción de un árbol de decisiones. En cada nodo del árbol se evalúa una variable y se realiza una división del conjunto de datos en dos o más ramas, de manera que se maximice la pureza de la información en cada rama. Este proceso se repite recursivamente hasta que se alcanza un criterio de parada predefinido.

La idea detrás de este modelo es que, al construir un árbol de decisiones, se pueden representar de manera explícita y visual las reglas de decisión que se aplican para llegar a una determinada predicción. Además, el árbol de decisiones permite identificar las variables más importantes para la predicción, y puede ser utilizado para explicar de manera intuitiva y comprensible el proceso de toma de decisiones.

En este proyecto, el modelo de árbol de decisión fue utilizado como base para desarrollar modelos más complejos, como *Random Forest* y *XGBoost*. Si bien el árbol de decisión es un modelo útil, no es suficientemente preciso para ser utilizado como modelo predictivo final en este proyecto. Por esta razón, no se muestran las métricas de rendimiento para este modelo en particular.

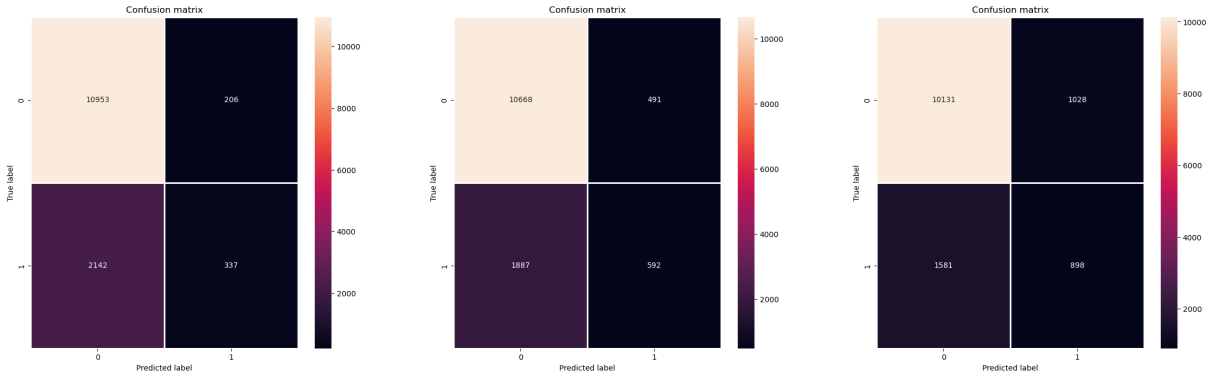
No obstante, la comprensión del árbol de decisiones es fundamental para entender cómo funcionan los modelos más complejos que se desarrollaron en este proyecto. En la documentación, se explicará en detalle cómo se utilizaron los modelos de *Random Forest* y *XGBoost* para predecir los atrasos en vuelos, y cómo se comparan con el modelo de árbol de decisiones en términos de rendimiento y precisión.

5.3. Random Forest

El *random forest* [3] es un modelo de *machine learning* basado en árboles de decisión. Consiste en crear múltiples árboles de decisión y entrenarlos con porciones del set de entrenamiento, entrenando a cada uno con solo algunas *features* y muestras. De esta manera a pesar de entrenar a cada árbol para que prediga la misma característica, cada uno es entrenado de manera diferente, lo que evita que los árboles estén muy correlacionados y aporta a que nuestro modelo pueda generalizar los datos. Finalmente los árboles ponen sus predicciones juntas y en base a estas el Random Forest elige el resultado

La configuración de hiperparámetros empleada para el modelo de *Random Forest* fue:

- **max depth:** 10
- **min samples leaf:** 1
- **min samples split:** 2
- **n estimators:** 200



(a) Sin balanceo (b) SMOTE (c) RUS

Figura 10: Matrices de confusión con el dataset sin balancear, con oversampling y con undersampling del random forest.

La Figura 10 muestra los resultados de clasificación del modelo de random forest sin la aplicación de técnicas de balanceo (a), con SMOTE (b) y con Random Undersampling (c).
Para el random forest, los mejores resultados fueron obtenidos al entrenar con el set de datos con random undersampling (RUS).

Los resultados obtenidos por este modelo en el set de test fueron un puntaje AUC-ROC de 0.747 y una accuracy de 0.8087, los cuales indican un buen funcionamiento

También para constatar que nuestro modelo no sufre de *overfitting* probamos nuestro modelo con los datos con los que lo entrenamos, lo cual nos dio un puntaje AUC-ROC de 0.836 y una accuracy de 0.778, con lo cual podemos ver que el modelo generalizó bien, al no tener una diferencia de puntaje AUC-ROC inusualmente alta.

5.4. XGBoost

El Modelo XGBoost [10] es un algortimo de *machine learning* donde se ensamblan arboles de decision que puede predecir la probabilidad de que algo suceda, en nuestro caso la variable objetivo es si habría o no un atraso. Los modelos Random Forest y XGBoost son muy similares ya sea por el uso de arboles o porque combinan distintos modelos para poder obtener otro con mejores resultados. Sin embargo la diferencia es que ambos arman los modelos y utilizan los datos de entrenamiento de distinta manera.

En nuestro caso para armar los modelos se utiliza el boosting que crea modelos débiles para combinarlos y hacer predicciones mas precisas, donde se corrigen todo error que pueda haber en las predicciones del anterior modelo. Esto ayuda mucho para poder obtener una precisión mas exacta en casos difíciles, mejorando gradualmente el rendimiento del modelo.

Además cuenta con una gran variedad de hiperparametros las cuales son variables que se seleccionan antes de entrenar al modelo y son muy útiles para los resultados obtenidos. Algunos hiperparametros utilizados en nuestro caso fue la cantidad de arboles, la profundidad del árbol (es decir la cantidad máxima de niveles que tiene desde la raíz hasta las hojas) y el learning rate o tasa de aprendizaje que sirve para controlar a que velocidad deseamos que el modelo aprenda con los datos de entrenamiento.

El rendimiento del modelo va a depender de los hiperparámetros, la calidad de los datos que se utilizan en este ya que si bien hacemos uso del un aprendizaje secuencial entre los arboles para prestar atencion al margen de error, tambien pueden suceder casos donde se sobreajuste (overfitting) el modelo o que los datos no nos permiten obtener una gran precision. Por ello para poder lograr nuestro objetivo, debemos tener en cuenta en no trabajar con una excesiva cantidad de datos, que los que tengamos sean de utilidad y se pueda hallar la mejor combinacion de parametros para que nuestro modelo sea lo mas preciso posible

Durante las pruebas, primero se optó por trabajar tanto con los datos del DataSet brindado por el cliente sobre los vuelos como los del DataSet de los datos externos, entre ellos clima y distancia. Se

eligieron determinadas variables como en el resto de modelos y se realizaron pruebas tanto con los datos en bruto como normalizados de diferentes maneras (SMOTE, RUS, ENN, entre otras).

Los hiperparametros fueron ajustados aleatoriamente en una primer instancia para ver el funcionamiento del modelo. Con esto, los valores de las métricas y ROC-AUC no eran los mejores claramente. Por ejemplo, la ROC-AUC tanto del dataset del train como del test rondaban entre 0.60 y 0.72 en el mejor de los casos.

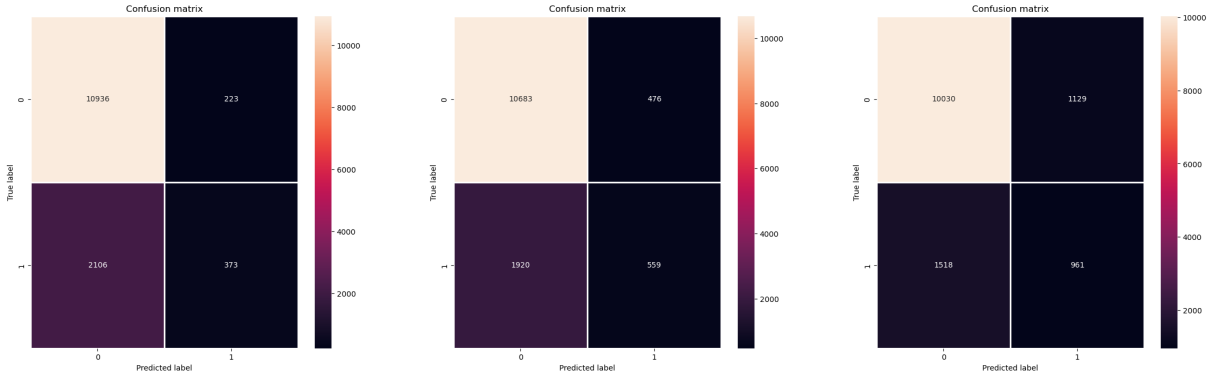
Luego de esto, se buscaron los mejores hiperparámetros para obtener mejores resultados. Para ello se armaron listas con determinados valores que fueron utilizados en el algoritmo de selección GridSearch [7], para así obtener diferentes resultados en base a las combinaciones de los hiperparámetros dados en las listas. Estos resultados se guardaron en un dataframe y se ordenaron según los mejores valores obtenidos en los datos separados para el test. Para poder apreciar de una mejor manera como afectaban estos hiperparámetros al modelo, se realizaron dos gráficos utilizando los mejores valores, de los que se obtuvieron algunas conclusiones. Por ejemplo, que a mayor cantidad de árboles, el resultado disminuía.

Una vez obtenidos los valores para los parámetros, se volvió a probar el modelo y se obtuvieron mejores resultados ya que ahora la ROC-AUC estaba entre 0.69 y 0.72 para los datos sin balancear como para los datos balanceados.

Estas métricas no se podían mejorar con los datos existentes. Sin embargo, con las features externas 'Promedio en minutos de atrasos anteriores' y 'Minutos desde el ultimo atraso' los valores incrementaron en buena medida, la ROC-AUC de los datos para el test alcanzaron un rango de valores entre 0.72 y 0.75 mientras que para los datos de train, alcanzaron un rango de 0.78 y 0.85.

La configuración de hiperparámetros empleada para el modelo de XGBoost fue:

- **max depth:** 6
- **objective:** binary:logistic
- **learning rate:** 0.02
- **n estimators:** 200



(a) Sin balanceo (b) SMOTE (c) RUS

Figura 11: Matrices de confusión con el dataset sin balancear, con undersampling y con oversampling del GXBoost.

Para el GXBoost, los mejores resultados fueron obtenidos al entrenar con el set de datos con random undersampling (RUS).Los resultados obtenidos por este modelo en el set de test fueron un puntaje AUC-ROC de 0.753 y una accuracy de 0.806, los cuales indican un buen funcionamiento

También para constatar que nuestro modelo no sufre de overfitting probamos nuestro modelo con los datos con los que lo entrenamos, lo cual nos dio un puntaje AUC-ROC de 0.794 y una accuracy de 0.749, con lo cual podemos ver que el modelo generalizó bien, al no tener una diferencia de puntaje AUC-ROC alta, sino solo una de unos pocos puntos.

5.5. CatBoost

CatBoost[1] es un algoritmo supervisado de aprendizaje automático que utiliza un método de Gradient Boosting Decision Tree (GBDT), lo cual es un ensamble de árboles de decisión implementados de forma iterativa con fin de disminuir el error de la predicción y aumentar su precisión, buscando minimizar u optimizar una función. En caso de ser un modelo de clasificación, se busca disminuir el valor de una función de perdida logarítmica (log loss); y en caso de tratar con un modelo de regresión, se intenta optimizar una función de error cuadrático medio (mean squared error).

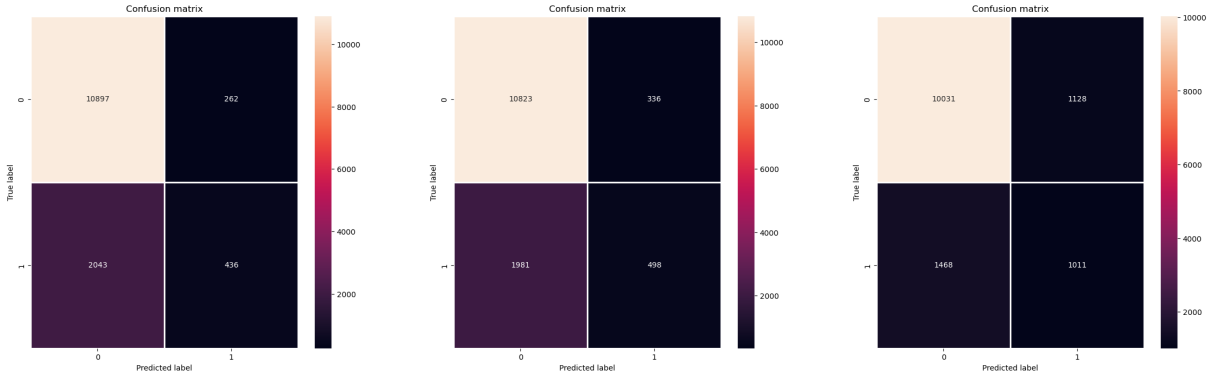
Una principal diferenciación entre el CatBoost y los demás algoritmos de GDBT es la utilización de árboles de decisión simétricos divididos en dos particiones, es decir que cada nodo de cada nivel del árbol tiene una hoja izquierda y otra derecha, dos particiones. Al poseer particiones simétricas, disminuye la probabilidad de que el modelo caiga en *overfitting*, y permite que no hayan grandes cambios en la magnitud de la calidad del modelo resultado cuando se varían los hiperparámetros. Facilitándonos su selección y brindándonos buenos modelos con sus valores predeterminados. Además suponiendo que el árbol tiene una profundidad h, la cantidad exacta de hojas del árbol va a ser 2^h, lo cual permite que el índice de cada hoja pueda ser calculado con operaciones binarias, obteniendo un procesado más eficiente de la CPU, por consecuente, menor tiempo de procesamiento y predicciones más rápidas.

Luego, se encuentra el manejo de variables categóricas sin la necesidad de un preprocesamiento previo. Por defecto CatBoost utiliza *one-hot-encoding*, pero también dispone de la combinación de variables, concatenándolas mediante un algoritmo voraz (greedy algorithm), y de un método llamado codificación ordenada (*ordered encoding*). Este se basa en la creación de una línea de tiempo artificial a partir de la permutación de los datos, donde a las variables categóricas se les da un valor numérico representativo a partir de un promedio de los valores anteriores al valor de interés categórico.

Por último, se encuentra el tipo de boosting que utiliza CatBoost. El *classical boosting*, entrena muchos modelos de árboles de decisión iterativamente, donde cada árbol corrige el error del anterior. El problema con este método es que, en cada iteración, los modelos se entrenan con el mismo conjunto de datos, lo que genera tendencias al *overfitting*. Entonces, lo que propone el *ordered boosting* es permutar los datos en cada iteración, para así entrenar los modelos con un conjunto de datos nuevo en cada etapa.

La configuración de hiperparámetros empleada para el modelo de CatBoost fue:

- **depth:** 6
- **iterations:** 400
- **learning rate:** 0.05
- **random seed:** 42



(a) Sin balanceo (b) SMOTE (c) RUS

Figura 12: Matrices de confusión con el dataset sin balancear, con undersampling y con oversampling del catboost.

Para el Catboost, los mejores resultados fueron obtenidos al entrenar con el set de datos con random undersampling (RUS).

Los resultados obtenidos por este modelo en el set de test fueron un puntaje AUC-ROC de 0.760 y una accuracy de 0.810, los cuales indican un buen funcionamiento

También para constatar que nuestro modelo no sufre de overfitting probamos nuestro modelo con los datos con los que lo entrenamos, lo cual nos dio un puntaje AUC-ROC de 0.801 y una accuracy de 0.759, con lo cual podemos ver que el modelo generalizó bien, al no tener una diferencia de puntaje AUC-ROC alta, sino solo una de unos pocos puntos.

5.6. Otras pruebas

5.6.1. Regresión Logística

El primer modelo que desarrollamos es la regresión logística [5], un modelo de aprendizaje automático que se utiliza para la clasificación binaria. Su objetivo es predecir la probabilidad de que un resultado pertenezca a una categoría determinada, en nuestro caso, a los vuelos atrasados.

La regresión logística se basa en la función sigmoide, que mapea cualquier valor a un rango entre 0 y 1. El modelo utiliza esta función para predecir la probabilidad de que un resultado pertenezca a la categoría deseada. Para estimar los parámetros que mejor se ajustan a los datos, el modelo usa la técnica de máxima verosimilitud.

Es un modelo muy efectivo en muchos casos, y tiene la ventaja de ser muy simple. Sin embargo, justamente por ser tan simple, encontramos que no era adecuado para nuestro proyecto debido a la naturaleza compleja de los datos y las relaciones no lineales entre las variables. El modelo falló a la hora de capturar estas relaciones complejas, dando un peor resultado. Por lo tanto, decidimos no incluirlo en nuestra presentación.

5.6.2. Multilayer Perceptron

Otro modelo que consideramos para nuestro proyecto fueron las redes neuronales, específicamente los multilayer perceptrons (MLP). Los MLP funcionan mediante la conexión de múltiples neuronas en diferentes capas, donde cada capa procesa información y la envía a la siguiente capa. Las entradas se propagan hacia adelante a través de la red hasta llegar a la capa de salida, donde se produce el resultado.

Los MLP son modelos flexibles y pueden adaptarse a diferentes tipos de problemas de clasificación y regresión. Para ajustar un MLP, primero se debe definir la arquitectura de la red neuronal, lo que incluye el número de capas ocultas, el número de neuronas en cada capa, las funciones de activación, entre otros. El modelo comienza con pesos aleatorios y luego utiliza un proceso iterativo llamado *backpropagation* para actualizar los pesos de la red neuronal y minimizar la función de pérdida en función de la diferencia entre los resultados esperados y los obtenidos.

Sin embargo, después de probar los MLP en nuestro conjunto de datos, descubrimos que los resultados no fueron mejores que los de los modelos que ya habíamos utilizado. De hecho, los MLP obtuvieron resultados similares, pero ligeramente peores que los modelos de *Random Forest* y *XGBoost*.

La razón principal de no utilizar MLP en nuestro proyecto fue que para poder ajustarlo y optimizarlo adecuadamente, probando distintas arquitecturas, tendríamos que dedicar mucho tiempo y recursos computacionales que exceden el alcance de nuestro proyecto. Además, dado que los otros modelos ya estaban dando resultados satisfactorios, no parecía justificable invertir tanto tiempo en una técnica que no mejoraría significativamente la precisión de las predicciones.

5.7. Ensambles

Los ensambles son una técnica de aprendizaje automático que combina múltiples modelos individuales para crear un modelo más poderoso y preciso. El objetivo de los ensambles es mejorar la precisión y la generalización de los modelos predictivos, reducir la varianza y mejorar la estabilidad de las predicciones.

El principio básico de los ensambles es que la combinación de varios modelos individuales puede ser más precisa que cualquier modelo individual. El objetivo de los ensambles es mejorar la precisión de los modelos predictivos y proporcionar resultados más robustos y confiables para la toma de decisiones en una amplia variedad de aplicaciones de aprendizaje automático.

Existen varios métodos de ensamble, donde utilizamos unos cuantos para poder medir y comparar métricas de nuestro proyecto. A continuacion se mencionan.

5.7.1. Stacking

El método de Stacking consiste en tomar los modelos que queremos tener como base, hacer predicciones con estos y entrenar a un nuevo meta modelo con estas predicciones para predecir el mismo target. Esto suele mejorar la precisión y, en especial, la generalización, pues se tienen en cuenta los puntos de vista de cada modelo con respecto a cada dato a predecir. La predicción final nos la da el meta modelo, teniendo como datos de entrada las predicciones de los modelos base.

Los resultados obtenidos con el modelo de stacking en nuestro caso fueron buenos, en especial la generalización, pero no mejoraron significativamente con respecto a otros modelos más simples.

5.7.2. Voting Classifier

El voting classifier es un método de ensamble que toma los modelos que nosotros le demos, los entrena sobre los mismos datos y luego predice haciendo un sistema de votación que puede ser dura, en la que cada modelo da una clase distinta y la que sea votada más es la elegida, o puede ser suave, donde los modelos dan la probabilidad de elegir una clase y a esas probabilidades se las promedia.

En nuestro caso los resultados dados por el voting classifier no fueron mejores a los obtenidos que otros modelos individualmente.

6. Conclusión

A partir de la premisa planteada, se trabajó en la construcción de diversos modelos de Machine Learning utilizando diversas técnicas de clasificación. Estos modelos se entrenaron y evaluaron con el objetivo de predecir si un vuelo iba a sufrir un retraso de más de 15 minutos o no, número que se utilizó para determinar si un vuelo había salido en horario o no.

Ejecutadas y analizadas todas las técnicas aplicadas al problema planteado de la predicción en el atraso de los vuelos, podemos afirmar que la selección de una u otra técnica ofrece resultados no tan dispares en este proyecto. En el caso particular de lo visto en este caso de uso, las distintas combinaciones de técnicas tanto de balanceo como sin balanceo no desvelan importantes diferencias en el rendimiento al tener en cuenta la respuesta final para la detección de si va a haber un atraso en la salida de los aviones.

Después de explorar varios modelos, hemos abordado diversas técnicas y herramientas para mejorar la capacidad predictiva de los modelos de clasificación sobre un conjunto de datos que incluía información sobre vuelos históricos, donde nos encargamos de crear una variable objetivo que pueda medir los atrasos.

Inicialmente se aplicaron algunos de los modelos de clasificación más comunes, como la Regresión Logística, Decision Tree, Random Forest, XGBoost y CatBoost. Posteriormente, se siguió explorando en busca de nuevas variantes, utilizando modelos como Multilayer Perceptron. Se exploraron algunas técnicas para mejorar estos modelos, como la selección de variables importantes y el ajuste de hiperparámetros.

Se discutieron técnicas específicas para cada modelo, como la importancia de las variables en Random Forest y XGBoost, la aplicación del GridSearchCV y RandomizedSearchCV para encontrar los mejores hiperparámetros en los modelos de clasificación. Además, se llevaron a cabo pruebas utilizando técnicas de Undersampling y Oversampling, así como la técnica de ensamblado para mejorar aún más el desempeño de los modelos.

Sobre la elección de un modelo específico, nos decantamos por el CatBoost ya que sus valores fueron superiores tanto en las métricas de precisión con un resultado de 0.809 y la curva AUC-ROC 0.761 valores superiores a las otras técnicas evaluadas. Cabe destacar que en nuestro caso particular el concepto de distancia entre observaciones no tenía un gran impacto en el rendimiento de los modelos, lo que podría explicar por qué el uso de la tecnica RUS (RandomUnderSampler), una técnica menos avanzada en comparación a otras de Undersampling, logró mejores resultados.

Finalmente, se llegó a la conclusión de que la combinación de diversas técnicas permitió mejorar la capacidad predictiva de los modelos, y se logró un valor de AUC-ROC de al menos 0,75 y una precisión de los modelos mayor a 0,80, objetivos que nos propusimos al inicio de la investigación mediante el uso de modelos como Random Forest, XGBoost y CatBoost.

Esto sugiere que estos modelos son capaces de predecir con precisión si los vuelos se retrasarán o no. Sin embargo, es importante tener en cuenta que se necesitó una cantidad significativa de prueba y error para llegar a estos modelos y asegurarse de que fueran óptimos. En general, se ha logrado el objetivo de encontrar modelos precisos para predecir los retrasos en los vuelos.

En resumen, la combinación de diversas técnicas y la exploración de múltiples modelos permitió obtener un mejor desempeño en la predicción de retrasos en vuelos, si bien seleccionamos el modelo CatBoost, cabe aclarar que todos alcanzaron los resultados esperados, lo que puede ser de gran utilidad para mejorar la eficiencia y la puntualidad en los viajes aéreos.

Referencias

[1] Catboost.
catboost.ai/en/docs/concepts/algorithm-main-stages.

[2] ENN. Imbalanced learn.
imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.EditedNearestNeighbours.

[3] Random Forest. Scikit learn.
scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.

[4] NCR. Imbalanced learn.
imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.NeighbourhoodCleaningRule.

[5] Logistic Regression. Scikit learn.
scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.

[6] RUS. Imbalanced learn.
imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.

[7] Grid Search. Scikit learn.
scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.

[8] SMOTE. Imbalanced learn.
imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.

[9] Decision Tree. Scikit learn.
scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.

[10] XGBoost. dmlc.
xgboost.readthedocs.io/en/stable/python/python_api.html.