Especialización en Ciencia de Datos

TRABAJO FINAL INTEGRADOR

PREDICCIÓN DE PROBABILIDAD DE ROBOS DE BICICLETAS DEL SISTEMA PÚBLICO DE LA CIUDAD DE BUENOS AIRES

Alumno: Jazmín Montero

Tutor: Dra. Leticia Gómez

Ciudad de Buenos Aires, Febrero 2022

Índice de Contenidos

1. Introducción 3
2. Contexto 5
3. Presentación del Problema 7
4. Justificación del Estudio 7
5. Alcances del Trabajo y Limitaciones 8
6. Objetivos 8
   - 6.1. Objetivo General 8
   - 6.2. Objetivos Específicos 8
7. Metodología Aplicada 9
   - 7.1. Modelo Conceptual 9
   - 7.2. Algoritmos a Utilizar 10
     - 7.2.1. Árbol de Decisión 10
     - 7.2.2. Regresión Logística 11
     - 7.2.3. Random Forest 12
   - 7.3. Métricas de evaluación 13
   - 7.4. Balanceo del Dataset 15
8. Desarrollo del Modelo 18
   - 8.1. Fuentes de Información 18
   - 8.2. Proceso de ETL Aplicado 18
   - 8.3. Análisis Descriptivo 20
   - 8.4. Categorización de los Viajes 24
   - 8.5. Preparación del Dataset 25
   - 8.6. Implementación de los Algoritmos 29
9. Resultados Experimentales 31
10. Conclusiones y Trabajo a Futuro 34
11. Referencias 36
12. Anexo 37

1. Introducción

Ecobici[1] es el sistema de bicicletas públicas disponible en la actualidad en la Ciudad de Buenos Aires y forma parte de una estrategia muy ambiciosa del gobierno local en lo que refiere a la movilidad sustentable. Este plan integra diversos programas desarrollados teniendo en consideración las mejores prácticas a nivel internacional y se apoya sobre cuatro pilares:

- Prioridad del transporte público
- Movilidad sustentable
- Ordenamiento del tránsito y seguridad vial
- Movilidad inteligente

Al igual que en muchas de las grandes ciudades del mundo, hubo y hay una demanda creciente de sendas exclusivas para la circulación de bicicletas. En particular, en la Ciudad de Buenos Aires, los primeros tramos corresponden al año 1996. Esta incipiente red, que originalmente sólo conectaba los barrios de Palermo y Belgrano se fue expandiendo con los años y creciendo en forma exponencial a partir del año 2004. En la actualidad, la misma alcanza más de 260 km, evolucionando, además, de simples sendas dedicadas a ciclovía protegida. La diferencia entre estos dos aspectos radica en que la segunda es un carril exclusivo para el uso de la bicicleta resguardado del resto del tránsito vehicular por medio de un separador físico.

Hay estudios indican que los porteños tienen más de un millón de bicicletas en sus hogares. Sin embargo, hasta hace pocos años, el uso de la misma como medio de transporte (seguro y eficiente) no estaba expandido. Ese problema es el primero que se intentó atacar con la creación y continua expansión de la red de ciclovías (dado que, en este caso, se entiende que el incremento de la oferta es lo que empuja la demanda principalmente). Por otro lado, se busca complementar con un servicio de bicicletas públicas y compartidas de manera de brindarle al usuario una mayor flexibilidad, pudiendo sacar y devolver la misma en cualquiera de las estaciones (y no tener así que cargar con ella).

El sistema público de bicicletas fue operado por el Gobierno de la Ciudad de Buenos Aires desde sus inicios. Sin embargo, en marzo del 2018, se reglamentó una ley que permite la gestión del mismo por parte de un tercero, cumpliendo ciertas condiciones de

[1] https://www.buenosaires.gob.ar/ecobici

inversión, mantenimiento y modernización. La empresa ganadora de la licitación pública fue Tembici, de origen brasilero y con explotación en ciudades como San Pablo, Río de Janeiro, Salvador, Pernambuco, Porto Alegre, Brasília, Vila Velha, Riviera (Brasil) y Santiago (Chile). El traspaso a la misma se dio a comienzos del año 2019. La transición implicó un desembolso de más de 70 millones de dólares, destinados a modernizar la totalidad del patrimonio (estaciones, anclajes y bicicletas), así como en la puesta a punto de los sistemas y equipos de mantenimiento. Por otro lado, hubo un cambio radical en la modalidad del sistema: se pasó de estaciones operadas por agentes públicos y con un horario diurno a estaciones autónomas, disponibles las 24hs los 365 días del año.

En abril de 2019 se habilitó el nuevo sistema y en el mes de junio se alcanzó el patrimonio objetivo: 400 estaciones y poco más de 4000 bicicletas. El servicio era totalmente gratuito hasta 1 hora de uso en días hábiles y 2 horas en fines de semana y feriados. Para poder alquilar un rodado, se requería completar un formulario de inscripción a través de una aplicación, pero no se efectuaba verificación alguna de la identidad de la persona.

Esto último, tuvo como consecuencia que hubiera numerosas suscripciones de usuarios con identidad desconocida, cuyo único objetivo era poder retirar una bicicleta de la estación y apropiársela. Esto, en adición a cierta falta de previsibilidad del sistema (rodados sin sensor GPS, carencia de cámaras de seguridad en puntos clave, entre otros), hizo que, hacia finales del 2019, el patrimonio se viera reducido a una cuarta parte de lo que era.

Tomando los datos de los más de 6,7 millones de viajes registrados en el sistema Ecobici en el año 2019, el presente trabajo se propone, mediante la utilización de un algoritmo de aprendizaje automático, estimar la probabilidad de hurto dadas las condiciones iniciales de un viaje: tipo de usuario (nuevo o recurrente), estación de origen, día/hora de la semana. De esta manera, se brinda una solución rápida (adicional a todas las otras medidas que se pueden implementar sobre el sistema) y que permitirá a los agentes del Gobierno de la Ciudad de Buenos Aires poder focalizar de una manera más eficiente los recursos, de forma de minimizar los hurtos y mejorar así la experiencia del usuario, a la vez que se disminuyen los costos y se hace más sostenible el sistema.

Un primer análisis de la información muestra que los robos suelen ser llevados a cabo por personas sin antigüedad en el sistema (o que crearon una cuenta justamente con esa intensión). En cuanto a las estaciones de origen, si bien los hechos están

bastante distribuidos, se observan algunos puntos más propensos. Por último, en cuanto al día y hora, se registra una mayor cantidad de robos en días viernes y en horas de la tarde.

Al igual que en los casos de estudio de transacciones fraudulentas con tarjeta de crédito, dado al desbalance en el set de datos (sobre más de 7 millones de viajes, sólo 3000 resultaron en hurto), resulta necesario aplicar técnicas de *sampleo* adecuadas, dado que, en caso contrario, cualquier algoritmo determinará que la probabilidad de hurto es 0 (o despreciable).

2. Contexto

El estudio del tipo de robo que aborda este trabajo es muy similar a todos los que se pueden consultar con respecto al uso de algoritmos de machine learning para detección de fraude, como por ejemplo en sistemas financieros, dado que comparten varias características:

- Se conocen las condiciones iniciales
- Dataset muy desbalanceado (los casos positivos representan menos del 1% del total)
- El tipo de error que se quiere evitar es de tipo II: si se considera fraudulenta una transacción o, análogamente, un viaje se presume que terminará en robo, y finalmente esto no sucede, el costo sobre el sistema es despreciable. Por ejemplo, en nuestro caso, sólo implicaría que un agente preste particular atención a dicho rodado y vea si es devuelto en tiempo y forma, a la vez que obtiene o rastrea los datos del usuario. Por el contrario, si marcamos un viaje como seguro y finalmente la bicicleta es robada, el costo es, como mínimo, el de reposición.

En cuanto al algoritmo a utilizar, se pueden testear distintos modelos, comenzando con uno sencillo (como por ejemplo, un árbol de decisión) y comparando con otros más complejos (por ejemplo, ensambles como *Random Forest*). En todos los casos, se obtienen las distintas métricas para comparar las bondades del modelo *testeado* a la vez que se evalúa algo determinante para nuestro caso: la velocidad de respuesta.

Lo determinante en nuestro caso, y tal como sucede con la detección de fraude, es el correcto balanceo del *dataset*. La mayoría de los algoritmos de aprendizaje

automático no funcionan de manera adecuada cuando el número de casos en cada una de las clases (en nuestro caso: Robada y No Robada) es muy desigual. Mismo tipo de situaciones se suelen encontrar en casos de estudios médicos (muestra de muchos pacientes con diagnóstico negativo y sólo unos pocos con caso positivo) o detección de fraude en transacciones (muchas transacciones con tarjeta de crédito, por ejemplo, en donde no hay problema alguno y pocas de ellas sí son fraudulentas).

Esto se debe a que la mayoría de los algoritmos están diseñados para maximizar la exactitud (número de aciertos sobre el total de registros) y reducir los errores (Kumar, 2020). Por lo tanto, se obtiene un valor muy alto para esta métrica (*accuracy*) únicamente prediciendo la clase mayoritaria, haciendo que se omita por completo la minoritaria (que es, justamente, para lo cual estamos creando un modelo de *machine learning*).

Para tratar de solucionar el problema del enorme desbalance del set de datos, tenemos dos opciones:

- Técnicas de *under-sampling*: se eliminarán registros de la clase mayoritaria de manera de que su número termine igualando, o sea muy aproximado, al de la clase minoritaria. En consecuencia, el tamaño del set de datos se reduce drásticamente, haciendo que los tiempos de cómputo sean mucho menores, en comparación con los datos originales.
- Técnicas de *over-sampling*: en el sentido inverso, buscar generar copias de los registros de la clase minoritaria de tal manera de llevarlas a un número que coincida (o sea muy aproximado) al de la clase mayoritaria. Es decir, el tamaño del *dataset* se duplicará, haciendo que los tiempos de cómputo para el entrenamiento de los algoritmos sean más altos.

Por supuesto, ninguna de las dos técnicas está libre de problemas. En el primer caso, el peligro es que se pierda información que pueda ser útil para la generación del modelo. En nuestro caso en particular, esto no parece ser, a priori, una amenaza significativa debido a la cantidad acotada de dimensiones que consideramos. En el segundo caso, el peligro puede ser el de sobre-ajustar (*overfitting*) el modelo.

Adicionalmente, al reducir en forma significativa el tamaño del dataset en el primer caso, el tiempo de cómputo y necesidad de almacenamiento se ven disminuidos. Lo contrario sucede en el caso de over-sampling.

Es por eso por lo que, en todos los modelos mediremos también el tiempo de cálculo requerido dado que esto puede ser un factor determinante al momento de la eventual implementación del mismo.

3. Presentación del Problema

En abril de 2019 se habilitó el nuevo método para el alquiler de un rodado en el sistema público de bicicletas en la Ciudad de Buenos Aires, en el cual se debía completar un formulario de inscripción a través de una aplicación, pero no se efectuaba verificación alguna de la identidad de la persona.

Esto último, tuvo como consecuencia que hubiera numerosas suscripciones de usuarios con identidad desconocida, cuyo único objetivo era poder retirar una bicicleta de la estación y apropiársela. Es así que, entre abril del 2019 y enero del 2020, más del 75% del patrimonio puesto a disposición en este sistema público fue robado. El ritmo en que desaparecían las bicicletas superaba con creces la velocidad posible de restitución de las mismas, a la vez que encarecía enormemente el costo de mantenimiento del sistema en su totalidad. En forma paralela, los habituales usuarios de las bicicletas veían cada vez menos rodados disponibles en las estaciones y la buena reputación que supo tener esta solución de movilidad en el arranque, se vio severamente afectada.

4. Justificación del Estudio

Además de la consecuencia directa, que es la del costo de reposición del rodado, existen también otras aristas que hacen necesario llevar adelante un estudio como este:

- Prácticas:
    - Destinar recursos y agentes a la búsqueda y recupero de los rodados
    - Evitar la pérdida de prestigio del sistema público de bicicletas
    - Posibilitar llevar adelante el Plan de Movilidad Sustentable tal como se lo previó.

- Burocráticas:
    - Varios de los componentes o partes con las que se arman las bicicletas son importados y los trámites aduaneros implican varias semanas de demora en la reposición
    - Necesidad de conseguir órdenes policiales/judiciales para poder intentar recuperar algunos rodados vistos (o sospechados) en casas particulares.

5. Alcances del Trabajo y Limitaciones

El presente estudio busca entrenar un modelo que permita predecir el grado de probabilidad de que una bicicleta termine siendo robada, dadas las condiciones iniciales del viaje.

Las conclusiones a las que se pretende arribar son consecuencia directa de la observación de los datos y no buscan, en ningún momento, estigmatizar de alguna manera a un tipo de usuario o alguna zona en particular de la ciudad.

Se trata de un algoritmo preliminar, en fase de desarrollo, y no de la implementación de un sistema completo de seguimiento, dado que no se tienen los accesos necesarios a los servidores del GCB.

6. Objetivos

6.1. Objetivo General

Desarrollar un modelo predictivo que permita establecer el grado de posibilidad de que un viaje termine con la bicicleta robada dadas las condiciones iniciales del mismo.

6.2. Objetivos Específicos

- Realizar un análisis descriptivo del uso de las bicicletas públicas de la Ciudad de Buenos Aires entre abril del 2019 y marzo del 2020.
- Establecer un criterio homogéneo por el cual se considerará a una bicicleta como *robada*.
- Depurar el set de datos y realizar el balanceo del mismo.
- Probar distintos algoritmos de aprendizaje automático de clasificación, con complejidad creciente.

- Obtener las métricas que permitan la comparación entre los distintos modelos.
- Determinar cuál es el algoritmo que mejor se ajusta al objetivo general planteado, dadas las métricas a evaluar (*accuracy, ROCAUC, F1,* etc.) así como el tiempo de cómputo necesario.

7. Metodología Aplicada

A los fines de mostrar una prueba conceptual piloto, implementaremos tres algoritmos predictivos sobre el mismo dataset balanceado, para determinar cuál es el que mejor se ajusta al objetivo general planteado.

7.1. Modelo Conceptual

El set de datos a utilizar únicamente contemplará las variables iniciales de un viaje, dado que el momento en que un usuario retira un rodado de una estación es cuando se quiere encender la alarma y no ante un posible hurto posterior.

Para ello, a partir del set original de datos, se adecuarán y obtendrán las siguientes dimensiones:

- Día de semana
- Hora del día
- Estación de origen
- Cantidad de viajes por usuario acumulada a la fecha
- Fecha del primer viaje por usuario
- Clasificación = Robada / No Robada

Con esos datos estaremos teniendo en cuenta información geográfica (de una manera u otra, todo está condensado en el dato de la estación de origen), temporal (día de la semana y hora) y del usuario. Luego, se intentará detectar si con ello es suficiente para asignar con cierta certeza, la posibilidad de que la bicicleta será robada.

Es importante señalar que podemos prescindir de la fecha como tal y sólo conservar el dato del día de la semana dado que, en este caso, no es de importancia la estacionalidad. Por supuesto, si se tuviera información de varios años y se observara

cierto patrón (por ejemplo, si se robaran más bicicletas al comienzo de la primavera), ahí sí se conservaría como dato adicional.

Con respecto al usuario, se entiende que contar con más información, por ejemplo, método de pago utilizado o nombre/apellido, aportaría mayor precisión a la predicción. Esto es porque es esperable que, si una bicicleta fue robada, entonces el registro sea apócrifo (nombre y/o apellidos inventados o uso de la cuenta de otra persona) o que haya diferencias de acuerdo al tipo de pago registrado al momento de crear la cuenta (tarjeta de crédito, débito o prepaga).

7.2. Algoritmos a Utilizar

Se probará, para cada uno de los algoritmos de *resampling* propuestos, tres algoritmos de aprendizaje automático que buscarán predecir la variable objetivo (en nuestro caso, si la bicicleta fue robada o no): árbol de decisión, regresión logística y random forest.

7.2.1. Árbol de Decisión

Es un algoritmo de aprendizaje automático supervisado en donde se van siguiendo reglas para clasificar un registro. Es decir, se usan los datos de entrenamiento de tal manera de generar, a partir de las distintas variables, preguntas que van dividiendo el dataset en dos regiones en forma sucesiva.

Cada vez que, en términos de una variable, se genera una nueva pregunta, se está creando un nodo de decisión. Los nodos que ya no se dividen porque son nodos puros en donde todos los registros son de la misma clase o porque simplemente no se pueden dividir más, dado que se configura una determinada profundidad de árbol, conforman las hojas del árbol. Cuando se llega a este último nivel, el algoritmo clasifica cada uno de los registros según el nodo al que pertenecen.

En cada split, el algoritmo trata de dividir el dataset en el menor subset posible, intentando minimizar la función de costo o pérdida que se haya establecido. En nuestro caso en particular, utilizaremos la Entropía (en lugar de la Impureza de Gini), en donde se mide el desorden de la agrupación de acuerdo a la variable objetivo (Robada / No Robada).

En términos generales, si no se establece un número máximo de niveles de decisión, el algoritmo intentará subdividir el dataset lo más posible, lo que conlleva el peligro de sobreajuste u overfitting. Por el contrario, un árbol muy pequeño correrá peligro de subajuste o underfitting (resultado en un sesgo muy significativo).

7.2.2. Regresión Logística

Es un método estadístico que permite predecir clases, en principio, binarias (cómo nuestro caso: Robada / No Robada). Aún siendo uno de los algoritmos más sencillos (junto con la regresión lineal), es uno de los más utilizados para resolver este tipo de problemas.

En términos generales, la regresión logística permite describir y estimar la relación entre una variable binaria dependiente y todas las restantes variables independientes. Se modela utilizando una función sigmoide, cuyo resultado es un valor cualquiera entre 0 y 1. Ver Figura 1.

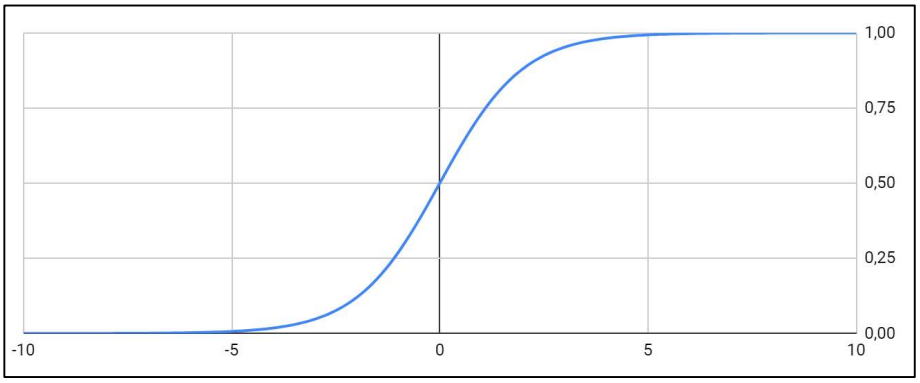


Fig. 1 – Función Sigmoide

A partir de las variables de entrada, se modelará una curva sigmoide cuyo resultado obtenido corresponde a la probabilidad de pertenecer a una clase (valor 0) o a la otra (valor 1). Es decir, si el resultado obtenido es mayor a 0.5, entonces corresponderá a la clasificación con valor 1.

Para ajustar la función sigmoide, se minimizará la suma de los cuadrados de la distancia entre los distintos registros y la curva ("*mean squared error*" como función de costo).

7.2.3. Random Forest

Los árboles aleatorios son uno de los métodos de ensamble más utilizados y se utilizan tanto para regresiones como para algoritmos de clasificación (como en nuestro caso). Estos métodos así se llaman, dado que involucran la utilización simultánea de varios modelos para mejorar así la performance. En particular, el resultado que arroje este método será una media ponderada entre el output de varios árboles de decisión individuales.

Lo que el algoritmo hace es lo siguiente:

- Se forman N segmentaciones aleatorias del set de datos (incluso, un subset puede contener más de una vez al mismo registro). Tampoco, necesariamente, tienen que ser del mismo tamaño (de hecho, a esto hace referencia la palabra Random en el nombre del algoritmo).
- Con cada subset aleatorio, se entrena un árbol individual. Entonces, al tener varios árboles, resulta en un método con el que se puede atacar el principal problema que tienen los árboles de decisión individuales, que es el overfitting.
- Al momento de hacer una predicción, se obtiene un valor determinado para cada árbol individualmente entrenado (en nuestro caso, cada árbol de decisión arrojará uno de los dos resultados posibles: Robada / No Robada).
- La clasificación que más veces se obtiene a lo largo de todo el bosque de árboles será, entonces, el resultado de la predicción del modelo para ese registro que se tomó para la predicción.

7.3. Métricas de evaluación

Aprovechando las métricas de la librería *sklearn*, definiremos:

- **Matriz de confusión**: en ella se observa cómo se clasifican los registros del *subset* de *testing* (Robada / No Robada) y se compara contra el valor real.

	Predicción - No Robada	**Predicción - Robada**
Real - No Robada	TN	FP
Real - Robada	FN	TP

Por supuesto, siempre se buscará en todo algoritmo de aprendizaje automático, maximizar los aciertos (TN + TP). Adicionalmente, en nuestro caso particular, siempre será preferible minimizar los Falsos Positivos (FP) dado que serán aquellos casos en los que se ha robado una bicicleta y lo hemos obviado. Esto acarrea un costo mayor al caso opuesto (predecir que será robada cuando en realidad no lo es). En un caso, como mínimo, se tiene el costo económico de reposición de la bicicleta. En el otro, será simplemente el costo del esfuerzo dedicado al seguimiento de dicho viaje (el algoritmo toma las variables de origen del viaje, predice que será robada y el agente del GCBA podrá hacer un seguimiento del usuario, el rodado, ajustar cámaras de seguridad, alertas a agentes de tránsito, etc.)

- **Accuracy**: es el porcentaje de casos que el modelo ha acertado.

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

- **Precision**: se corresponde con el número de positivos verdaderos (TP) con respecto al total de aciertos (TP + FP). Es decir, una baja precisión será sinónimo de un alto número de falsos positivos.

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

- **Recall**: este indicador (también llamado Exhaustividad o Sensitivity) será un indicador de la cantidad que el modelo es capaz de identificar. Un valor bajo denotará un número alto de falsos negativos. En consecuencia, en nuestros modelos buscaremos maximizar este valor (para evitar predecir como No Robadas aquellas que fueron, efectivamente, Robadas).

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

- **F1 Score**: es un promedio ponderado (media armónica) entre los dos anteriores (Precision y Recall)

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- **Área bajo la curva ROC (ROCAUC):** expresa probabilidad de que nuestro modelo distinga correctamente entre las observaciones de las 2 clases.
  Si se grafica la tasa de positivos verdaderos (TP) contra la de falsos positivos (FP), se obtiene la curva ROC correspondiente a nuestro modelo. Cuanto más se acerca dicha curva hacia el borde izquierdo-superior, mejor será la capacidad de nuestro modelo para distinguir entre ambas clases (es decir, el área bajo la curva corresponde a 1). Naturalmente, valores de tasa de TP coincidentes con la tasa de FP (línea a 45º), se corresponde con una clasificación totalmente aleatoria (área bajo la curva de 0,5). En la Figura 2 se puede ver la curva ROC mencionada.

Fig. 2 – Curva ROC

En estos términos, entonces, se suelen establecer los siguientes criterios generales:

- ROCAUC = 0,5 → modelo inútil / clasificación aleatoria
- ROCAUC < 0,7 → performance sub-óptima
- ROCAUC < 0,8 → buena performance
- ROCAUC > 0,8 → excelente performance
- ROCAUC = 1 → clasificación perfecta

Al trabajar con sets de datos desbalanceados, el valor de la métrica Accuracy será muy poco representativo de la bondad del modelo (se puede tener una alta tasa de aciertos simplemente prediciendo 0 valores positivos - es decir, aplicado en nuestro caso, que nunca se robarán ninguna bicicleta).

En consecuencia, veremos el total de las métricas en su conjunto, prestando particular atención al porcentaje de falsos negativos obtenidos, el valor de Recall y el área debajo de la curva ROC.

7.4. Balanceo del Dataset

En detalle, se estudiarán y aplicarán las siguientes técnicas de sampleo, todas ellas disponibles en la librería de Python *imblearn*, que será nuestra principal herramienta:

- **Undersampling**

  - Random undersampling: selección aleatoria de casos de tal forma de bajar el número de registros de la clase mayoritaria hasta balancear el dataset.

  - Near Miss: este algoritmo calcula la distancia entre todos los puntos de la clase mayoritaria y aquellos de la minoritaria, para luego eliminar aquellos de la mayoritaria que más lejos están de la otra clase y resultar así en un set de datos balanceado. En la Figura 3 se esquematiza el resultado del algoritmo.

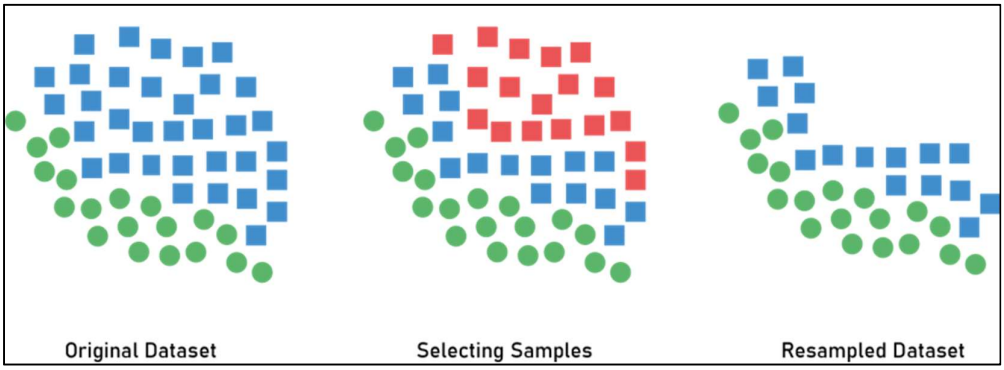


Fig. 3 – Algoritmo Near Miss (Charfaoui, 2021)
Fuente: Charfaoui, Y. (2021)

- Tomek Links: el algoritmo busca aquellos pares de registros compuestos por un elemento de cada clase que estén más próximos, para luego eliminar, únicamente de estos pares, los registros de la clase mayoritaria. Esto no genera un dataset balanceado (50/50), pero sí uno en donde las clases están más separadas que en el original. Se puede ver el resultado esquematizado del algoritmo en la Figura 4.

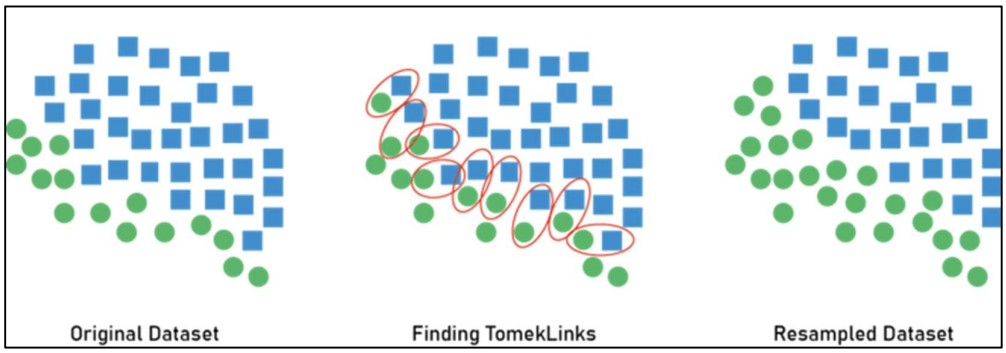


Fig. 4 – Algoritmo Tomek Links (Charfaoui, 2021)
Fuente: Charfaoui, Y. (2021)

- **Oversampling**

  - Random oversampling: al igual que en el caso de undersampling, se seleccionarán al azar elementos de la clase minoritaria, generando copias hasta alcanzar un dataset balanceado.
  - Synthetic Minority Oversampling Technique (SMOTE): este algoritmo selecciona al azar puntos de la clase minoritaria y computa los k vecinos más cercanos. Luego, se generan copias o registros sintéticos de la clase minoritaria entre los puntos seleccionados y sus vecinos. En la Figura 5 se esquematiza el resultado del algoritmo.

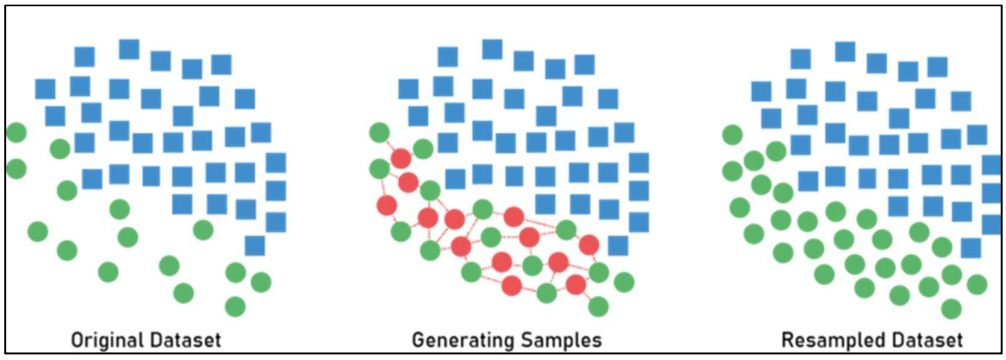


Fig. 5 – Algoritmo SMOTE (Charfaoui, 2021)
Fuente: Charfaoui, Y. (2021)

  - Adaptative Syntetic Oversampling (ADASYN): genera muestras de la clase minoritaria basándose en su distribución de densidad. El algoritmo genera datos para la clase robada (en nuestro caso) que serán más difíciles de aprender. Para ello, se miden los k vecinos más cercanos para todas las instancias de la clase minoritaria, se calcula la proporción de clases y en función de esto último, se generan los nuevos registros. En la práctica, termina siendo similar al anterior (SMOTE), pero generará diferente número de muestras sintéticas dependiendo del estimado de la distribución local de la clase minoritaria.

8. Desarrollo del Modelo

8.1. Fuentes de Información

Se hizo un pedido de información pública al Gobierno de la Ciudad de Buenos Aires, solicitando los datos de los viajes realizados a través del sistema público de bicicletas entre enero 2019 y marzo 2020, considerando las siguientes dimensiones:

- Id de usuario
- Fecha/hora de comienzo del viaje
- Fecha/hora de finalización del viaje
- Estación de origen
- Estación de destino
- Id del rodado

Para ello, se utilizó el sistema de formulario disponible en la web del GCBA[2]. Los datos fueron solicitados, en la medida de lo posible, en formato de texto plano.

8.2. Proceso de ETL Aplicado

El lenguaje utilizado para el procesamiento de los datos, su modelado y preparación para desarrollar los algoritmos predictivos y el análisis de sus resultados fue Python. Para ello, se confeccionó un notebook en Google Colab.

El archivo de origen es de texto plano (.txt) con sus campos separados por una tabulación. Haciendo una vista preliminar, se observaron múltiples campos que no serían necesarios y por ello, sólo se importaron las columnas indicadas a continuación:

- ID_Viaje: identificador único para cada una de las transacciones / viajes realizados
- Estado_cerrado: detalla el status del viaje (uno completado normalmente tiene el estado “NORMAL”).
- Fecha_Inicio: fecha-hora a la que se retira una bicicleta de una estación
- Nombre_Inicio_Viaje: nombre de la estación de origen
- Fecha_Final: fecha-hora a la que se devuelve una bicicleta en una estación

[2] https://www.buenosaires.gob.ar/gobierno/informacion-publica

- Nombre_Final_Viaje: nombre de la estación final
- ID_de_ciclista: identificador único del usuario
- Msnbc_de_bicicleta: identificador único del rodado

Para el trabajo con el dataframe se utilizó la librería Pandas, mientras que, para el trabajo con los campos de fecha y hora, la utilizada fue datetime.

El archivo de origen (viajes_ecobici_.txt), contiene 8.431.508 registros, con información de todos los viajes realizados entre 2019 y 2020. En la Figura 6 se pueden ver las características.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8431508 entries, 0 to 8431507
Data columns (total 8 columns):
 #   Column               Dtype 
---  ------               ----- 
 0   ID_Viaje             int64 
 1   Estado_cerrado       object
 2   Fecha_Inicio         object
 3   Nombre_Inicio_Viaje  object
 4   Fecha_Final          object
 5   Nombre_Final_Viaje   object
 6   ID_de_ciclista       int64 
 7   Msnbc_de_bicicleta   object
dtypes: int64(2), object(6)
memory usage: 514.6+ MB
```

Fig. 6 – Características del dataset

Lo primero que se observó al recorrer el set de datos es que, para los campos *Fecha_Inicio* y *Fecha_Final* teníamos registros en un formato tal que la librería *Pandas* los interpretaría como tipo *datetime*, mientras que otros estaban en formato de tiempo UNIX.

Por lo tanto, lo primero que se hizo fue separar el set de datos en dos partes, de acuerdo con el formato de las fechas. Utilizando funciones de la librería *datetime*, se pudo homogeneizar los formatos de los 4 campos (fecha de inicio y fin para cada una de las 2 partes) y se volvió a consolidar la totalidad de los registros en un único *dataframe*. En la Figura 7 se puede consultar el fragmento de código correspondiente.

```
import datetime as dt

fechas_unix['Fecha_Inicio'] =
pd.to_datetime(fechas_unix['Fecha_Inicio'], unit = 's')
fechas_unix['Fecha_Final'] =
pd.to_datetime(fechas_unix['Fecha_Final'], unit = 's')

fechas_dt['Fecha_Inicio'] = pd.to_datetime(fechas_dt['Fecha_Inicio'])
fechas_dt['Fecha_Final'] = pd.to_datetime(fechas_dt['Fecha_Final'])
data = pd.concat([fechas_dt, fechas_unix])

del fechas_dt
del fechas_unix

data['Fecha_Inicio'] =
pd.to_datetime(data['Fecha_Inicio'], utc = True)
data['Fecha_Final'] =
pd.to_datetime(data['Fecha_Final'], utc = True)

data = pd.DataFrame(data)
```

Fig. 7 – Fragmento de código para tratamiento de fechas

El primer paso, como se detalló previamente, fue conservar únicamente aquellos registros que resultaban de interés para el análisis. Estos son aquellos comprendidos entre abril del 2019 (cuando entró en vigor el servicio) y marzo del 2020 (hasta que se desató la pandemia y el sistema fue cerrado en forma temporal).

Por otro lado, también fue necesario eliminar aquellos registros cuyos campos para el msnbc de la bicicleta era nulo (NaN), así como para el campo ID_Viaje. En este caso, la cantidad de registros eliminados no superó el 0,01% del dataset.

En el Anexo se pueden consultar los detalles de la implementación.

8.3. Análisis Descriptivo

Lo primero que quisimos evaluar es la evolución del número de viajes en el tiempo. Para ello, hicimos un recuento de los registros agrupándolos por fecha (de inicio de viaje), y utilizando la librería matplotlib, pudimos obtener una visualización de esta distribución, la cual puede consultarse en la Figura 8.

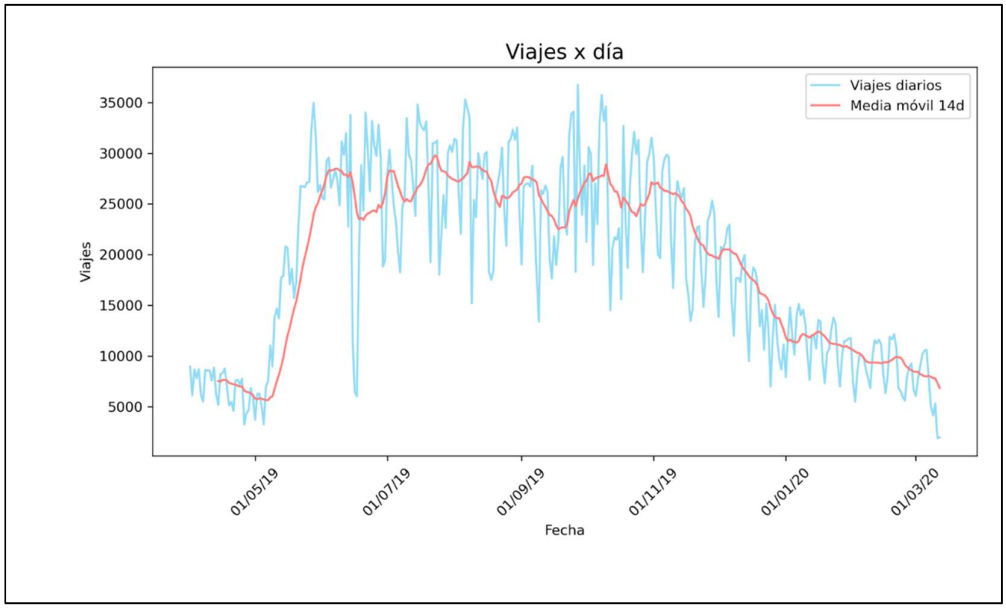


Fig. 8 – Distribución de cantidad de viajes en el tiempo

Se pudo observar que a partir del mes de mayo del 2019 y con la incorporación del total de la flota de rodados, se alcanzó un nivel estable en torno a los 25 / 30 mil viajes diarios. Esta situación se mantuvo relativamente pareja hasta mediados de octubre, en donde el declive comenzó hasta que finalmente, el sistema fue dado de baja con el comienzo de las medidas restrictivas de la cuarentena.

Una premisa a considerar era que cada día, toda bicicleta disponible para su uso, fuera usada al menos una vez. Se entiende que la posibilidad de que esto no sucediera (es decir, que una bicicleta en buen estado permanezca en una estación sin ser alquilada a lo largo de una jornada) era muy baja. Para comprobar esto, así como analizamos los viajes por día, hicimos lo mismo contando la cantidad de rodados disponibles a diario y efectuamos el cociente entre ambos valores. En la Figura 9 se puede ver la distribución obtenida.

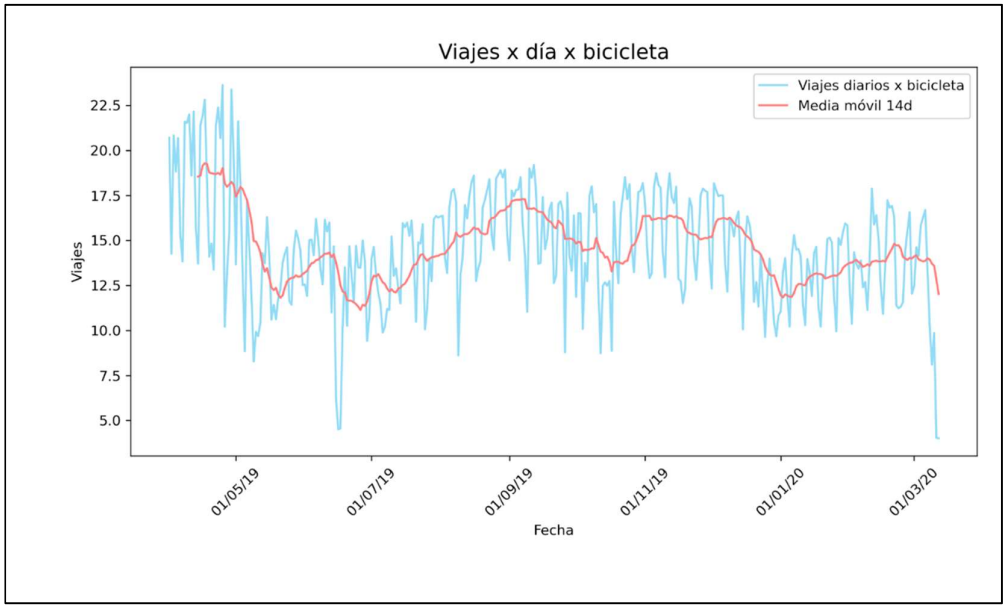
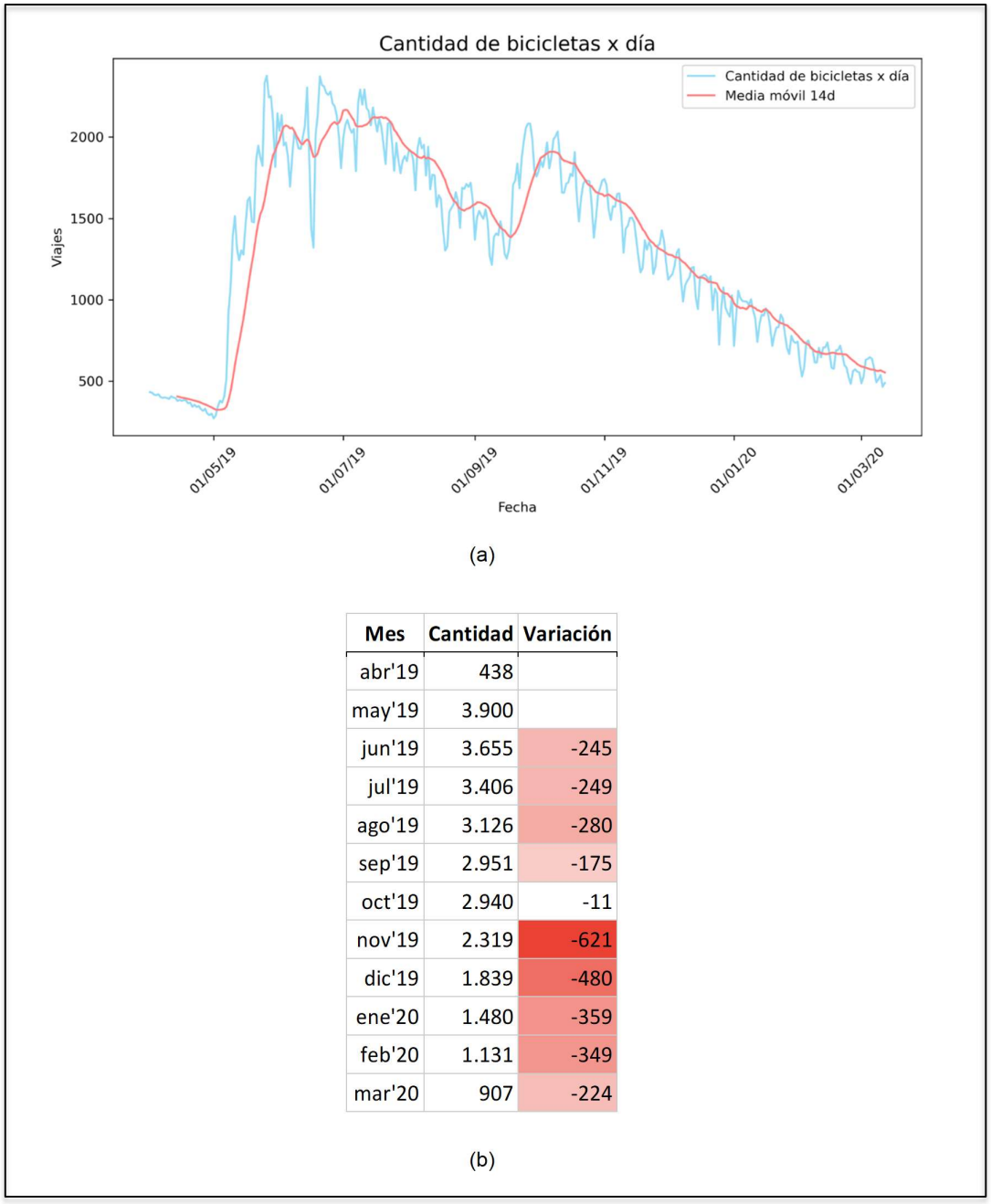


Fig. 9 – Distribución de cantidad de viajes por rodado en el tiempo

Se pudo observar que el promedio para la totalidad del período era de 14,6 viajes por bicicleta por día. Con este resultado, pudimos asegurar que la presunción realizada (que toda bicicleta disponible es utilizada a diario) era acertada. Esto resultó muy importante, dado que nos acercaba a la premisa que utilizamos posteriormente: “una bicicleta que deja de tener viajes (dada una determinada ventana de tiempo) implica que, o bien se sacó del sistema intencionalmente (no estaba en condiciones de circular) o fue robada”.

Por último, se analizó la cantidad de bicicletas disponibles por día (haciendo el recuento distintivo sobre el campo Msnbc_de_bicicleta) y se obtuvo su distribución. En la Figura 10 se puede consultar dicha distribución.



(a)

Mes	Cantidad	Variación
abr'19	438	
may'19	3.900	
jun'19	3.655	-245
jul'19	3.406	-249
ago'19	3.126	-280
sep'19	2.951	-175
oct'19	2.940	-11
nov'19	2.319	-621
dic'19	1.839	-480
ene'20	1.480	-359
feb'20	1.131	-349
mar'20	907	-224

(b)

Fig. 10 – Distribución de cantidad de rodados disponibles en el tiempo:
(a) distribución por día; (b) distribución acumulada por mes

La gráfica en función del tiempo permite apreciar cómo en el mes de octubre hubo una nueva inyección de rodados en el sistema para compensar la pérdida que venía ocurriendo desde julio. Sin embargo, en menos de un mes esta reposición fue

neutralizada y el número de bicicletas disponibles fue cayendo en forma progresiva, consecuencia de los robos. De un máximo en torno a 3900 rodados, alcanzamos un mínimo por debajo de las 1000 unidades antes de que se cerrara el sistema en marzo del 2020.

En el Anexo se pueden consultar el código fuente que se aplicó para obtener las distintas distribuciones.

8.4. Categorización de los Viajes

Tomando la premisa previamente presentada (toda bicicleta que se deja de usar fue, con alta probabilidad, robada), se procedió a categorizar cada uno de los registros del dataset en dos grupos: viaje que terminó correctamente y viaje que terminó con la bicicleta robada. Para ello, se propuso lo siguiente:

1. Se consideraron únicamente los viajes únicamente hasta fin de febrero del 2020 (para evitar tomar los datos de marzo del 2020, en donde progresivamente la gente fue resguardándose en sus hogares y, por lo tanto, el comportamiento y el uso del transporte público fue bastante anómalo).
2. Se ordenó el set de datos por el identificador del rodado (msnbc) y fecha crecientemente.
3. Se obtuvo la última fecha en la cual cada rodado efectuó un viaje. Si dicha fecha era anterior a la última fecha considerada (marzo 2020) y dada una determinada ventana (30 días, para ser conservadores), entonces se consideró que el rodado fue robado en el último viaje que efectuó.

De esta forma, la variable “Robada” toma valor verdadero (True) cuando se cumplan las condiciones antedichas. Si efectuamos una suma simple sobre la misma (para Python, True = 1), obtenemos un total de 3.221 registros que terminaron en robos. Esto es bastante acorde a lo esperado.

Adicionalmente, evaluamos esta variable (Robada), en función del tiempo. En la Figura 11 se puede ver la distribución obtenida.

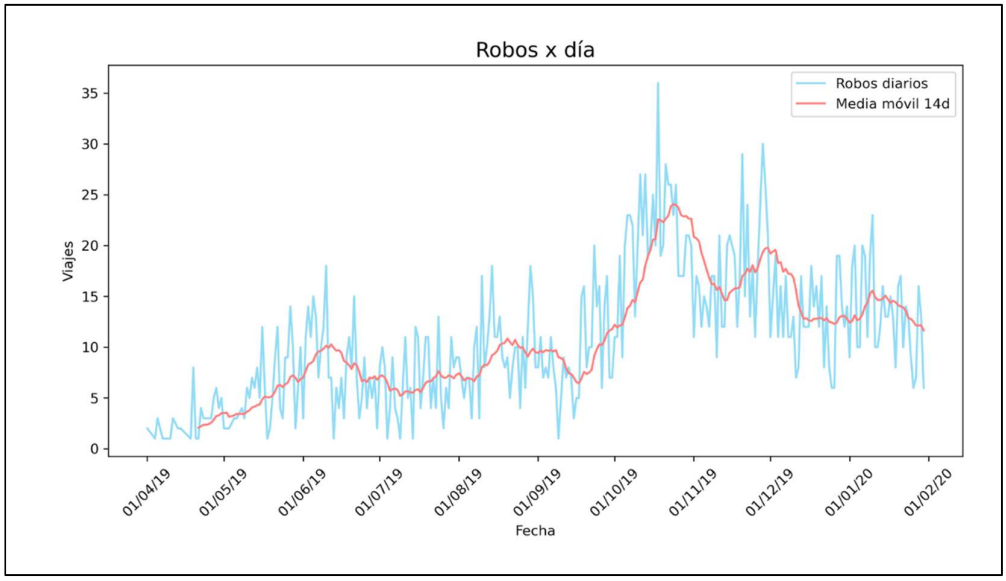


Fig. 11 – Distribución de rodados sustraídos en el tiempo.

El promedio para el período considerado arrojó un valor cercano a **11 bicicletas robadas por día**. El día en que más rodados se robaron fue el 18 de octubre de 2019, con un total de 36 unidades.

En el Anexo se pueden consultar el código fuente que se aplicó para obtener la categorización de los viajes.

8.5. Preparación del Dataset

Implementada la variable anterior, que es aquella que buscamos que el algoritmo prediga, se preparó el set de datos para correr los distintos algoritmos de aprendizaje automático.

Para ello, se armó una copia del dataframe original, conservando las columnas que pueden resultar de interés para evaluar la posibilidad de un nuevo robo a futuro:

- Día de semana
- Hora del día
- Estación de origen
- Cantidad de viajes por usuario acumulada a la fecha
- Fecha del primer viaje por usuario

Con el dataset listo, y antes de proceder con los algoritmos para intentar predecir el estado de la variable *Robada* (True/False), se realizaron algunos análisis descriptivos adicionales sobre esta misma variable, para estudiar *a priori* el efecto o el peso de las variables que consideradas.

Se comenzó estudiando los robos por antigüedad del usuario y por cantidad de viajes acumulados. En las Figuras 12 y 13 se pueden ver las distribuciones obtenidas.

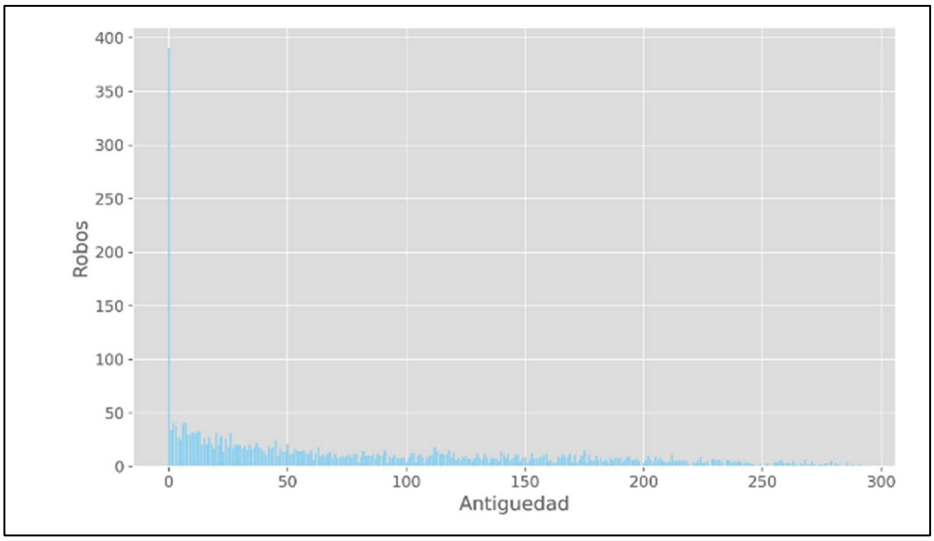


Fig. 12 – Distribución de robos por antigüedad del usuario.

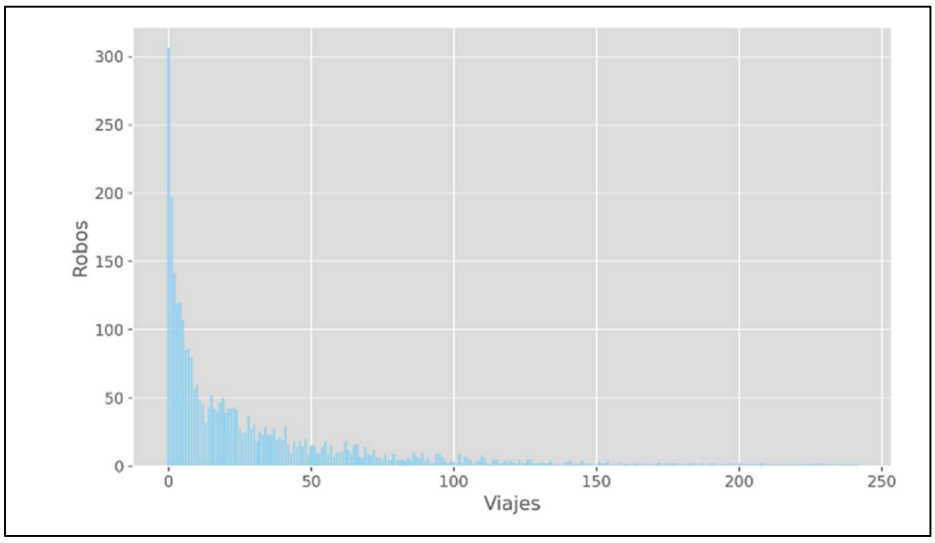


Fig. 13 – Distribución de robos por cantidad de viajes acumulados.

A simple vista se pudo observar que esta variable tendría una gran importancia en la predicción del robo o no de un rodado, dado que, en un gran porcentaje de los casos, los mismos son llevados a cabo por usuarios nuevos en la plataforma y con ningún o muy pocos viajes acumulados. Es decir, lo más probable es que una bicicleta sea robada por un usuario nuevo, cuya cuenta se crea especialmente para este fin.

También estudiamos la distribución de robos en función de la estación de origen en donde se tomó el rodado. En la Figura 14 se muestra el resultado.

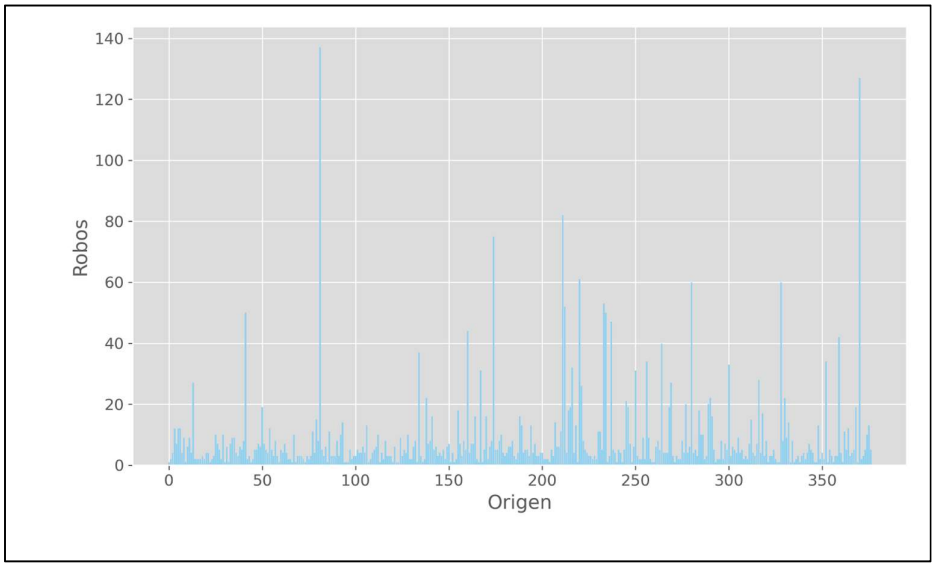


Fig. 14 – Distribución de robos por estación de origen.

En términos de la estación de origen, se calculó la distribución de robos en las más de 400 estaciones (no hay relación alguna entre el número categórico asignado y la posición geográfica real). Se pudo ver que, en aproximadamente 350 estaciones hubo 20 robos o menos. Es decir, las estaciones que sufrieron hurtos en reiteradas ocasiones eran relativamente pocas y, por lo tanto, también *a priori* pudimos prever que esta variable sería un buen predictor.

Finalmente, se analizó la cantidad de robos por día de la semana y hora del día. Las distribuciones obtenidas pueden visualizarse en las Figuras 15 y 16.

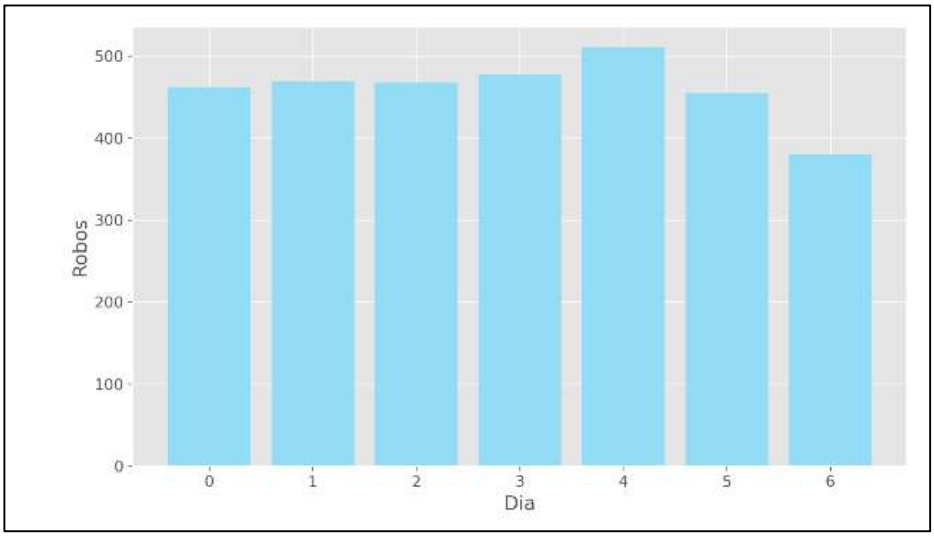


Fig. 15 – Distribución de robos por día de la semana (0 es Lunes).

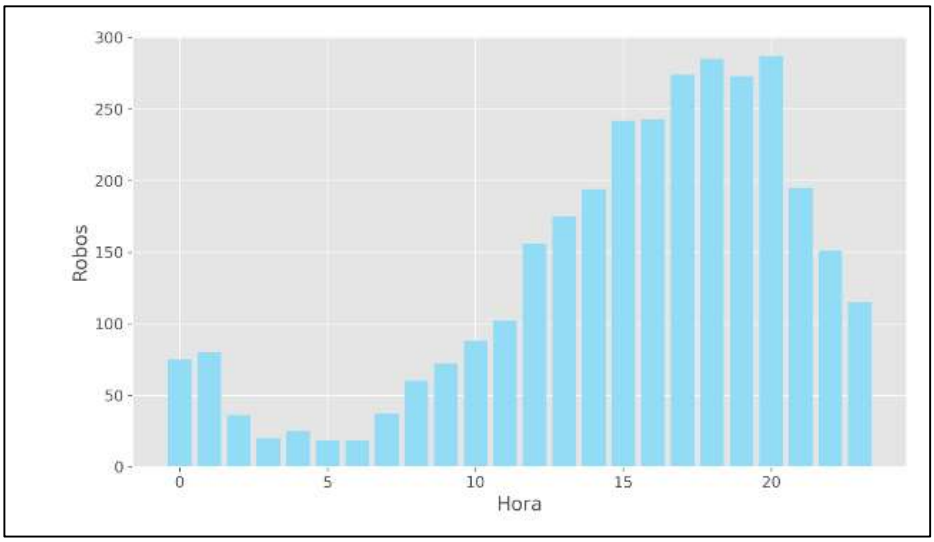


Fig. 16 – Distribución de robos por hora del día.

En cuanto al día de la semana, los robos estaban distribuidos en forma relativamente homogénea, siendo el viernes aquel que presentaba el valor máximo de ocurrencias. Por otro lado, en el desglose horario, se evidenciaba que los casos comenzaban a incrementarse recién superado el mediodía, siendo los valores máximos aquellos en torno a las 18 y 20hs. En conclusión, pudimos prever que la hora del día sería un buen driver, pero no así el día de la semana.

En el Anexo se pueden consultar el código fuente de todo lo realizado en esta Sección.

8.6. Implementación de los Algoritmos

El primer paso fue transformar las columnas del *dataframe* en listas, de manera de poder pasarlas como argumentos a los distintos modelos.

La librería de Python que usamos para implementar los distintos modelos fue *sklearn*. El set de datos se separó en los conjuntos de entrenamiento (70%) y *testing* (30%), de forma aleatoria.

El primer modelo aplicado, a modo de patrón de comparación sin balanceo del dataset, fue Árbol de Decisión, especificando una profundidad máxima de 4 niveles, y utilizando el criterio *entropy* para la separación en los nodos:

Un primer valor para evaluar rápidamente el modelo fue medir la exactitud (o accuracy), que nos indica qué tan frecuentemente la clasificación es correcta. En este caso, obtenemos un valor de 99,9%. Si bien, a priori, uno podría pensar que es algo positivo, la realidad es que significa que el modelo no nos está diciendo nada. Esto es consecuencia de la enorme disparidad que hay entre las clases del set de datos, en donde tan solo poco más de 3000 viajes terminaron en robos dentro de un universo de casi 7 millones de transacciones.

Para clarificar este concepto, construimos la matriz de confusión. En el caso del árbol de decisión entrenado, obtuvimos la matriz que se muestra en la Figura 17.

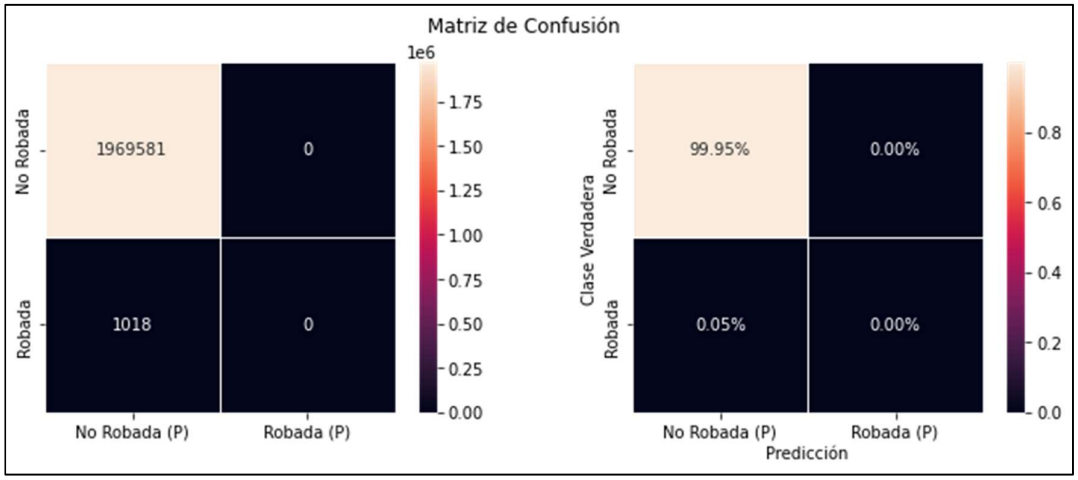


Fig. 17 – Matriz de Confusión para árbol de decisión inicial (izq: sin normalizar; der: normalizada).

El modelo, así como estaba originalmente, predecía que habría 0 robos de bicicletas. Esto es natural, debido a que el modelo estaba muy desbalanceado: menos del 0,05% de los viajes resultaba en robo y, por lo tanto, para este (y cualquier otro modelo), el resultado más esperado era que el viaje no termine nunca en robo del rodado.

Lo que se hizo a continuación, fue probar cada uno de los métodos de rebalanceo del dataset propuestos (6 en total), comenzando primero con aquellos de undersampling (Random, NearMiss y TomekLinks) y finalizando con los de oversampling (Random, SMOTE, ADASYN). Cabe destacar que, al trabajar con una mayor cantidad de registros, los modelos demoraron mucho más en ser entrenados.

Para cada uno de dichos métodos, además, se evaluaron los 3 algoritmos propuestos: Árbol De Decisión, Regresión Logística y Random Forest, obteniendo así un total de 18 resultados experimentales, sumados al del árbol de decisión entrenado con el dataset original (desbalanceado).

En el Anexo se pueden consultar el código fuente del undersampling y del oversampling aplicados a cada uno de los tres modelos predictivos.

9. Resultados Experimentales

En esta Sección mostramos los resultados obtenidos al aplicar el modelo de Árbol de Decisión sobre el dataset desbalanceado y los tres modelos Árbol de Decisión, Regresión Logística y Random Forest sobrea el dataset balanceado, tanto con los métodos Random, NearMiss y TomekLinks de undersamplig, como con los métodos Random, SMOTE y ADASYN de oversamping.

En Figura 18 se pueden ver los resultados resumidos para los 19 modelos entrenados en una computadora con procesador Intel i7, 32Gb de RAM y disco SDD. Los mismos fueron ordenados en forma decreciente por el indicador de área bajo la curva ROC (ROCAUC).

Lo primero a observar es como prácticamente todas las alternativas superan en algún aspecto al al resultado obtenido con respecto al patrón de comparación (set de datos original, sin balanceo, entrenado con un algoritmo de árbol de decisión), en donde prácticamente no hay distinción alguna contra una clasificación aleatoria, dado que al área bajo la curva ROC es 0.5.

Acorde al objetivo que se persigue, se deben ponderar más aquellos resultados en donde los indicadores de Recall y ROCAUC se maximicen. Eso debido a que, no sólo se busca que el modelo distinga correctamente entre ambas categorías (maximizar ROCAUC) sino también, que el número de falsos negativos obtenidos sea mínimo, ya que queremos evitar clasificar como “No Robada” a una bicicleta que sí lo fue.

En dichos términos, el algoritmo Random Forest entrenando al dataset con oversampling (total aproximado de 12 millones de registros) obtiene las mejores métricas, siendo la técnica de sampleo aleatoria la que mejor funciona para este modelo.

Sin embargo, al guiarnos por el tiempo requerido de entrenamiento, observamos valores en torno a los 2800 segundos (esto supera los 45 minutos de ejecución). Allí encontramos una diferencia radical con el resultado obtenido por el método NearMiss de undersampling. En este último caso, el set de datos se reduce a menos de 7000 registros y eso hace que al ejecutar el algoritmo de Random Forest, sólo demore 170 milisegundos, con una diferencia en el Recall y ROCAUC muy poco significativa.

Método	Accuracy	Precision	Recall	ROCAUC	F1-Score	Tiempo(seg)
RF - Random OS	0,99961	0,99923	1,00000	0,99961	0,99961	2608,74
RF - ADASYN OS	0,99567	0,99174	0,99966	0,99567	0,99568	2893,61
RF - SMOTE OS	0,99567	0,99174	0,99966	0,99567	0,99568	2899,29
RF - NearMiss US	0,98086	1,00000	0,96134	0,98067	0,98029	0,17
AD - NearMiss US	0,94982	0,99548	0,90452	0,95017	0,94782	0,01
RL - NearMiss US	0,93947	0,99528	0,88192	0,93891	0,93518	0,03
AD - SMOTE OS	0,62249	0,62216	0,62363	0,62249	0,62290	27,85
AD - ADASYN OS	0,62139	0,62166	0,62117	0,62139	0,62141	28,50
RF - Random US	0,61666	0,63130	0,59736	0,61706	0,61386	0,39
AD - Random OS	0,61690	0,60744	0,66225	0,61687	0,63366	26,88
AD - Random US	0,60942	0,60244	0,65879	0,60908	0,62936	0,00
RL - ADASYN OS	0,57946	0,57536	0,60851	0,57944	0,59147	31,71
RL - SMOTE OS	0,57940	0,57538	0,60832	0,57937	0,59139	34,47
RL - Random OS	0,57775	0,57212	0,61561	0,57776	0,59307	32,13
RL - Random US	0,57734	0,56812	0,63589	0,57749	0,60010	0,03
AD - Raw dataset	**0,99948**	**1,00000**	**0,00000**	**0,50000**	**0,00000**	**12,08**
RL - Tomek US	0,99951	1,00000	0,00000	0,50000	0,00000	20,52
AD - Tomek US	0,99951	1,00000	0,00000	0,50000	0,00000	12,97
RF - Tomek US	0,99949	0,00000	0,00000	0,49999	0,00000	1027,21

Referencias:

- AD = Árbol de decisión
- RL = Regresión Logística
- RF = Random Forest
- OS = Oversampling
- US = Undersampling

Fig. 18 – Tabla de resultados del AdD sobre dataset original, y de los tres modelos entrenados con dataset balanceado con undersampling y oversampling

Como última comparación, observemos en particular las matrices de confusión para los dos modelos de mejor rendimiento, Random Forest con oversampling Random y con undersampling NearMiss, que se pueden visualizar en las Figuras 19 y 20, respectivamente.

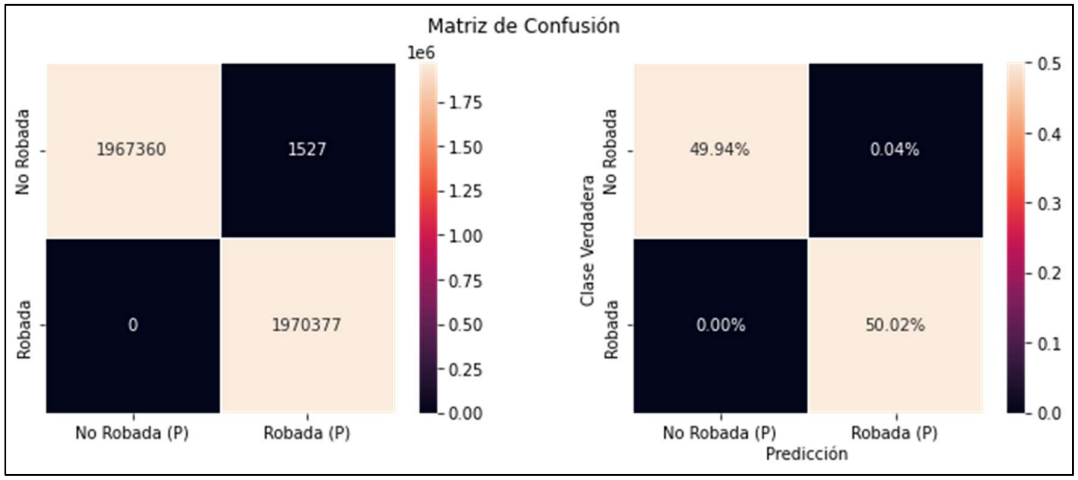


Fig. 19 – Matriz de Confusión para Random Forest con dataset balanceado por oversampling Random (izq: sin normalizar; der: normalizada).

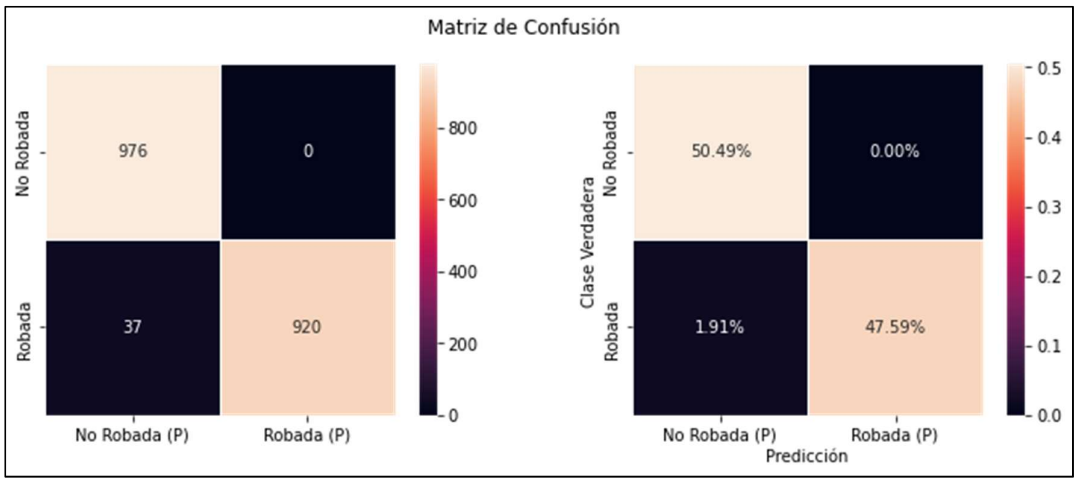


Fig. 20 – Matriz de Confusión para Random Forest con dataset balanceado por undersampling NearMiss (izq: sin normalizar; der: normalizada).

Más allá de que ambos modelos clasifican muy bien ambas categorías (99,96% en el primer caso vs. 98,08% en el segundo, considerando los casos TP + TN), el primer modelo, aunque tarde más en entrenar, tiene 0 casos en los que un viaje se haya predicho como No Robada cuando efectivamente lo fue. Esto, a los fines prácticos del costo de ambos tipos de errores, es primordial.

En consecuencia, para terminar de definir con qué algoritmo y método de balanceo es conveniente predecir en el caso real del sistema de alquiler de rodados, habría que definir cómo se implementá el script en producción y con qué frecuencia sería necesario re-entrenar el modelo con nuevos datos. Para esto último, la variable a evaluar será si hay cambios importantes en el sistema de alquiler de las bicicletas. Por ejemplo, con qué frecuencia hay apertura de nuevas estaciones, ampliación sustancial de la flota, cambio en los requerimientos de registro de los usuarios, validación de las identidades, etc. Con una la frecuencia de re-entrenamiento quincenal o mensual, no sería problemático tener un modelo que demore 45 minutos o más en ser entrenado. Cabe aclarar que, una vez completado el entrenamiento, obtener la predicción para un nuevo registro es prácticamente instantáneo. Por el contrario, si fuera crucial contar con un tiempo de entrenamiento de milisegundos, entonces el método de undersampling NearMiss, en conjunto con el algoritmo Random Forest, será la mejor opción, a riesgo de obtener un mínimo porcentaje de falsos positivos en predicciones de bicicletas que no serán robadas.

En el Anexo se pueden consultar las 18 matrices de confusión obtenidas al aplicar los seis métodos de balanceo por undersampling y oversampling a los tres modelos predictivos.

10. Conclusiones y Trabajo a Futuro

En abril de 2019, el Gobierno de la Ciudad de Buenos Aires habilitó un nuevo método de alquiler de rodados, con el sólo registro de un formulario de inscripción a través de una aplicación, sin mayor verificación de identidad. Como consecuencia, se incrementó notablemente la suscripción de usuarios con identidad desconocida, cuyo único objetivo era poder retirar una bicicleta de la estación para sustraerla.

En este trabajo se realizó un estudio con varios métodos de predicción y varios métodos de oversamping y undersampling para balancear el conjunto de datos originales, con el fin de proponer el mejor modelo predictivo que permita establecer el grado de posibilidad de que una bicicleta termine siendo robada, dadas las condiciones iniciales del viaje.

Si bien se trabajó en la determinación de un algoritmo preliminar, en fase de desarrollo, y no de la implementación de un sistema completo de seguimiento, los resultados obtenidos con muchos de los modelos fueron excelentes en cuanto a performance de clasificación. Cabe mencionar que, en prácticamente todos los casos, se utilizaron las opciones de configuración por defecto de cada uno de los métodos de sampleo y algoritmos de aprendizaje automático.

De todas maneras, es importante destacar que el modelo propuesto como el mejor, puede ser ampliamente enriquecido. Por ejemplo, incorporando información de los usuarios en cada uno de los viajes (método de pago, banco emisor, cantidad de transacción rechazadas, indicador de identidad validada, etc.), información de la estación de origen (presencia de cámaras de seguridad, cercanía a comisarías, cantidad de bicicletas robadas histórica, indicador de presencia regular de agentes viales, etc.) o del rodado mismo (antigüedad de éste, estado de mantenimiento, GPS activado y funcional, etc.).

Como trabajo a futuro, queda investigar si al incorporar más información al modelo propuesto, mejora la predicción obtenida. Para ello, será mandatorio hacer un ajuste fino de los distintos parámetros que ofrecen tanto el método de sampling como el algoritmo. A modo de ejemplo, se puede trabajar con distintas profundidades en elcaso del árbol de decisión, y testear si este factor mejora los resultados respecto del modelo elegido.

Por otro lado, también hay técnicas para explorar desde el lado de la Ciencia de Datos propiamente dicha. Por ejemplo, hacer validación cruzada en lugar de la simple división entre set de entrenamiento y set de testeo, o correr los algoritmos con un parámetro variable para dicha separación (en nuestro caso, fija en 70% entrenamiento + 30% testeo).

A modo de cierre, en este trabajo se alcanzando muy buenos resultados en cuanto a la posibilidad de predecir si un rodado será robado o no, únicamente considerando las variables de inicio de cada viaje. Esto puede ser una herramienta muy útil para el GCBA, de fácil y rápida implementación, que podría mejorar el uso de los recursos y mantenimiento del sistema, así como la satisfacción general de los usuarios.

11. Referencias

Alencar, R. (2017). *Resampling Strategies for Imbalanced Datasets*. Obtenido de Kaggle: https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

Avallone, M. (2019). *B Invasión Bicicleta*. Obtenido de https://binvasionbicicleta.com.ar/2019/02/26/breve-resena-de-los-sistemas-de-bicicletas-publicas-en-el-mundo/

Charfaoui, Y. (2021). *Resampling to Properly Handle Imbalanced Datasets in Machine Learning*. Obtenido de DEV Community: https://dev.to/charfaouiyounes/resampling-to-properly-handle-imbalanced-datasets-in-machine-learning-4anb

Gobierno de la Ciudad de Buenos Aires. (s.f.). *¿Qué es el plan de Movilidad Sustentable?* Obtenido de https://www.buenosaires.gob.ar/movilidad/que-es-el-plan-de-movilidad-sustentable

Gobierno de la Ciudad de Buenos Aires. (s.f.). *Pedaleá la Ciudad*. Obtenido de https://www.buenosaires.gob.ar/ecobici/pedalea-la-ciudad

Haibo H., Yunqian M. (2013). *Imbalanced Learning: Foundations, Algorithms, and Applications.* John Wiley & Sons Inc.New Jersey, 2013.

Hartshorn, S. (2016). *Machine Learning With Random Forests And Decision Trees: A Visual Guide For Beginners.* Kindle edition, 2016.

James, G., Witten, D., Hastie, T., Tibshirani, R (2013), *An Introduction to Statistical Learning with Applications in R,* Springer, 2013

Mckinney, W. (2017), Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, 2nd edition, O'Reilly Media, Inc, USA, 2017.

Tan, P., Steinvbach, M., Kumar, V. (2015), *Introduction to Data Mining,* Pearson Addison Wesley, 2005

Severance, C. R., Blumenberg, S, Hauser, E., Andrion, A. (2016). *Python for Everybody: Exploring Data in Python 3.* CreateSpace Independent Publishing Platform, 2016.

12. Anexo

Link Google Colab: https://cutt.ly/wUmQvt2

```
from google.colab import drive
import pandas as pd
import datetime as dt

# Importo el archivo
drive.mount('/content/gdrive')
campos = ['ID_Viaje',
          'Estado_cerrado',
          'Fecha_Inicio',
          'Nombre_Inicio_Viaje',
          'Fecha_Final',
          'Nombre_Final_Viaje',
          'ID_de_ciclista',
          'Msnbc_de_bicicleta']

file = "/content/gdrive/My Drive/ECD/viajes_ecobici_.txt"

data = pd.read_csv(file,
                   delimiter = '\t',
                   encoding= 'cp1252',
                   usecols = campos,
                   low_memory = False)

# Separo los elementos con fecha unix de aquellos que no
fechas_unix =
pd.DataFrame(data.loc[data['Fecha_Inicio'].astype(str).str.len() ==
10])
id_min = min(fechas_unix.index)
fechas_dt = pd.DataFrame(data.loc[data.index < id_min])

# Tengo que corregir esos últimos datos de fechas que están en unix
(vs los primeros que están en datetime)
fechas_unix['Fecha_Inicio'] =
pd.to_datetime(fechas_unix['Fecha_Inicio'], unit = 's')
fechas_unix['Fecha_Final'] =
pd.to_datetime(fechas_unix['Fecha_Final'], unit = 's')

fechas_dt['Fecha_Inicio'] =
pd.to_datetime(fechas_dt['Fecha_Inicio'])
fechas_dt['Fecha_Final'] = pd.to_datetime(fechas_dt['Fecha_Final'])
data = pd.concat([fechas_dt, fechas_unix])
del fechas_dt
del fechas_unix

data['Fecha_Inicio'] = pd.to_datetime(data['Fecha_Inicio'], utc =
True)
data['Fecha_Final'] = pd.to_datetime(data['Fecha_Final'], utc =
True)
data = pd.DataFrame(data)

# Me quedo únicamente con los registros entre abril 2019 y marzo
2020 (inclusive)
data = data[(data['Fecha_Inicio'] >= '2019-04-01') &
(data['Fecha_Inicio'] < '2020-04-01')]
```

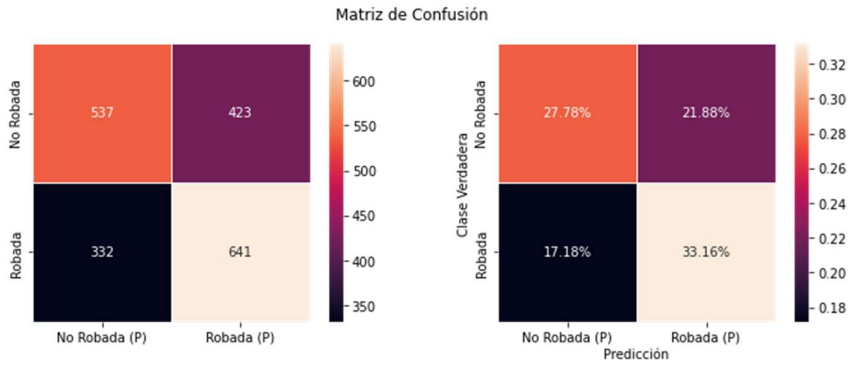
```
# Limpio los registros con ID y Msnbc erróneo
data.dropna(subset=['ID_Viaje', 'Msnbc_de_bicicleta'], inplace =
True)

# ----------------------------------------------------------------
# Acá iría todo código para el análisis descriptivo. Se deja el
contenido en el notebook de Colab
# ----------------------------------------------------------------

data.sort_values(by = ['Msnbc_de_bicicleta', 'Fecha_Inicio'],
                 ignore_index = True,
                 inplace = True)

fecha_max = '2020-03-01 00:00:00'
data.drop(data[data.Fecha_Inicio >= fecha_max].index,
          inplace=True)
data.reset_index(inplace = True,
                 drop = True)

# Incorporo una columna auxiliar con el valor del msnbc de la fila
siguiente
# De esta manera, cuando en una fila no coincidan, sabré que se
trata del último viaje para esa bicicleta

data['Msnbc_Siguiente'] = data['Msnbc_de_bicicleta'].shift(periods =
-1)
data['Ultimo_Viaje'] = data.eval('Msnbc_de_bicicleta !=
Msnbc_Siguiente')

fecha_max_data = max(data.Fecha_Inicio)
ventana = 30
margen = dt.timedelta(days = ventana)

fecha_max_final = fecha_max_data - margen

data['Robada'] = (data['Ultimo_Viaje'] == True) &
(data['Fecha_Inicio'] < fecha_max_final)

# Preparación de los datos para correr modelo ML

# Creo la copia
df = data.copy()

# Elimino las columnas que no me interesarán de entrada
df.drop(['ID_Viaje',
         'Estado_cerrado',
         'Fecha_Final',
         'Nombre_Final_Viaje',
         'Msnbc_de_bicicleta',
         'Msnbc_Siguiente',
         'Ultimo_Viaje'],
        axis='columns', inplace=True)

df.reset_index(inplace = True, drop = True)
```

```
# Ordeno los datos por ID ciclista y fecha, de manera de poder
incorporar la info del primer viaje, así como el acumulado
df.sort_values(by = ['ID_de_ciclista', 'Fecha_Inicio'],
               ignore_index = True,
               inplace = True)

df['ID_de_ciclista_anterior'] = df['ID_de_ciclista'].shift(periods =
1)
df['Primer_Viaje'] = df.eval('ID_de_ciclista !=
ID_de_ciclista_anterior')

# Incorporo una variable con cada viaje para poder contarlos
df['Contador'] = 1

# Incorporo una columna con la cantidad de viajes acumulados por
ciclista
df['Viajes_por_ciclista'] = df.groupby(by =
['ID_de_ciclista'])['Contador'].cumsum()
df.reset_index(inplace = True, drop = True)

# Incorporo una columna con la fecha del primer viaje de cada
ciclista
# Esto lo podemos usar como proxy del momento de registro
# Luego puedo calcular la antigüedad de cada usuario en el sistema

df['Fecha_primer_viaje'] = df.groupby(by =
['ID_de_ciclista'])['Fecha_Inicio'].cummin()
df['Antiguedad_ciclista'] = (df['Fecha_Inicio'] -
df['Fecha_primer_viaje']).dt.days

# Ahora, a partir de la fecha de inicio, creo columnas con el valor
correspondiente al día de la semana y la hora

df['Dia_semana'] = df.Fecha_Inicio.dt.weekday + 1 # De esta forma,
lunes = 1 // domingo = 7
df['Hora'] = df.Fecha_Inicio.dt.hour

# Por último, asigno un id a las estaciones (para evitar los strings
en los modelos de ML)
df = df.assign(ID_Origen =
(df['Nombre_Inicio_Viaje']).astype('category').cat.codes)

# Guardo en un dataframe el nombre de cada estacion con su ID (para
poder reasignar eventualmente)
estaciones = df[['Nombre_Inicio_Viaje', 'ID_Origen']]
estaciones = estaciones.drop_duplicates()
estaciones.reset_index(inplace = True, drop = True)

# Limpio las columnas auxiliares y aquellas que no me interesan y ya
queda listo el df para entrenar
df.drop(['Fecha_Inicio',
         'ID_de_ciclista_anterior',
         'Primer_Viaje',
         'Contador',
         'Fecha_primer_viaje',
         'Nombre_Inicio_Viaje'],
        axis='columns',
        inplace=True)
df.reset_index(inplace = True, drop = True)
```

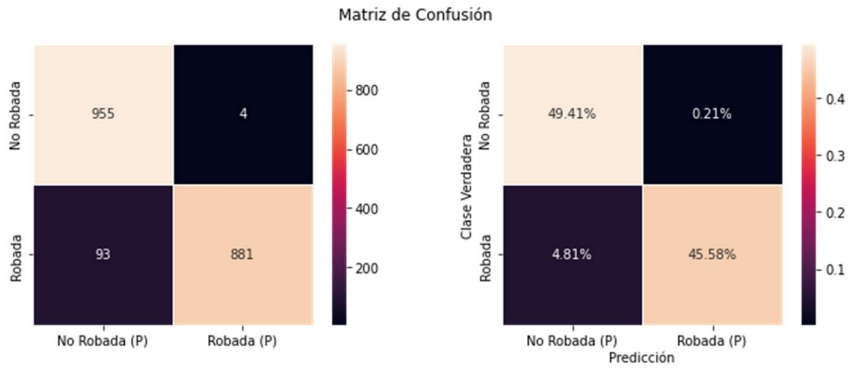
```
orden_columnas = ['Antiguedad_ciclista',
                  'Viajes_por_ciclista',
                  'ID_Origen',
                  'Dia_semana',
                  'Hora',
                  'Robada']
df = df.reindex(columns = orden_columnas)

df.rename(columns = {'Antiguedad_ciclista': 'Antiguedad',
                     'Viajes_por_ciclista': 'Viajes',
                     'ID_Origen': 'Origen',
                     'Dia_semana': 'Dia',
                     'Hora': 'Hora'},
          inplace=True)

import time
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC

from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN

from collections import Counter

# Preparación de los datos
y = list(df['Robada'])
X = list(zip(list(df['Antiguedad']),
             list(df['Viajes']),
             list(df['Origen']),
             list(df['Dia']),
             list(df['Hora'])))

# Separación del subset de entrenamiento y test
t_size = 0.3
# Entrenamos con el 70% de los datos al modelo
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size =
t_size)

modelo = DecisionTreeClassifier(criterion = 'entropy', max_depth =
4)

# Entrenamos...
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
```

```
# Testeamos...
y_pred = modelo.predict(X_test)

def matriz_confusion(y_test, y_pred):
    from sklearn.metrics import confusion_matrix
    #from sklearn.metrics import classification_report
    #import numpy as np
    import seaborn as sns

    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))

    # No Normalizada
    conf_matrix = confusion_matrix(y_test, y_pred)
    sns.heatmap(conf_matrix,
                xticklabels= ('No Robada (P)', 'Robada (P)'),
                yticklabels = ('No Robada', 'Robada'),
                annot=True, fmt='d',
                ax=ax[0],
                square=1, linewidth=1.);

    # Normalizada
    conf_matrix = conf_matrix.astype('float') / conf_matrix.sum()
    sns.heatmap(conf_matrix,
                xticklabels= ('No Robada (P)', 'Robada (P)'),
                yticklabels = ('No Robada', 'Robada'),
                annot=True, fmt='.2%',
                ax=ax[1],
                square=1, linewidth=1.);

    plt.suptitle('Matriz de Confusión')
    plt.ylabel('Clase Verdadera')
    plt.xlabel('Predicción')

    plt.show()

# Creo DF para ir guardando los resultados
column_names = ["Método",
                "Accuracy",
                "Precision",
                "Recall",
                "ROCAUC",
                "F1-Score",
                "Tiempo"]

resultados_df = pd.DataFrame(columns = column_names)

# Guardo los resultados
def guardo_resultados(metodo, y_test, y_pred, tiempo_ejecucion):
    from sklearn import metrics

    resultados = {}

    resultados['Método'] = metodo
    resultados['Accuracy'] = metrics.accuracy_score(y_test, y_pred)
    resultados['Recall'] = metrics.recall_score(y_test, y_pred)
    resultados['Precision'] = metrics.precision_score(y_test,
y_pred, zero_division = 1)
    resultados['ROCAUC'] = metrics.roc_auc_score(y_test, y_pred)
    resultados['F1-Score'] = metrics.f1_score(y_test, y_pred)
```

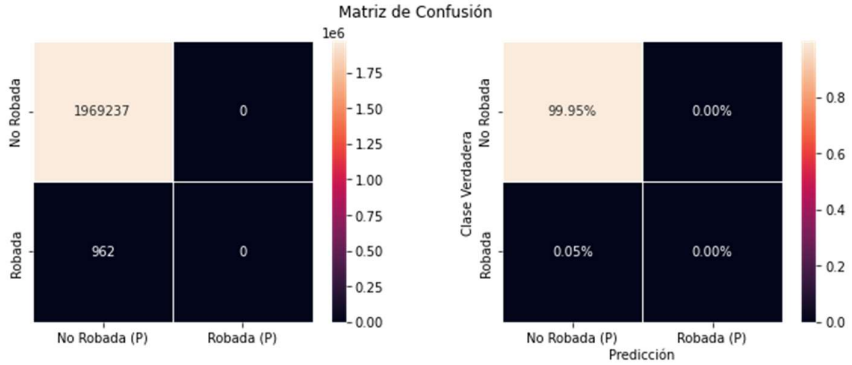
```
    resultados['Tiempo'] = tiempo_ejecucion

    return resultados

resultados = guardo_resultados('Árbol de decisión - Raw dataset',
y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)

# Establezco una semilla para todos los procesos aleatorios
seed = 142857

# UNDER SAMPLING
## Random Under-Sampling
rus = RandomUnderSampler(random_state = seed,
                         replacement=True)
x_rus, y_rus = rus.fit_resample(X, y)

## Under-sampling: NEAR MISS
nm = NearMiss()
x_nm, y_nm = nm.fit_resample(X, y)

## Under-Sampling: Tomek Links
tl = TomekLinks()
x_tl, y_tl = tl.fit_resample(X, y)

## Corremos los 3 modelos con los 3 métodos de undersampling
# Árbol de decisión
modelo = DecisionTreeClassifier(criterion = 'entropy', max_depth =
4)


# Random Under Sampling
X_train, X_test, y_train, y_test = train_test_split(x_rus, y_rus,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Árbol de decisión - Random Under-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```



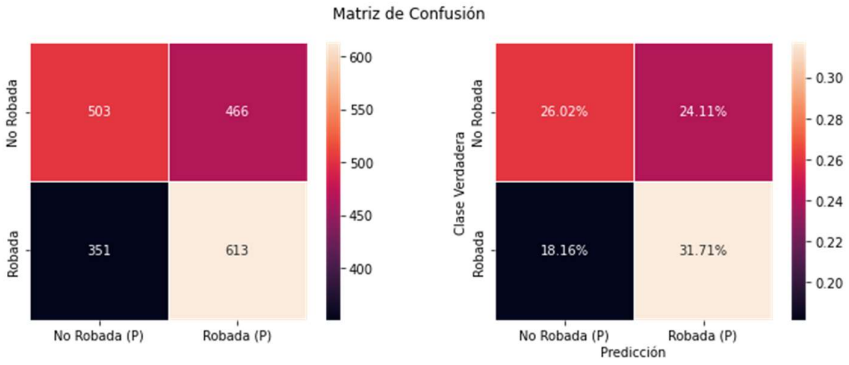
```
# Near Miss
X_train, X_test, y_train, y_test = train_test_split(x_nm, y_nm,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Árbol de decisión - NearMiss Under-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```



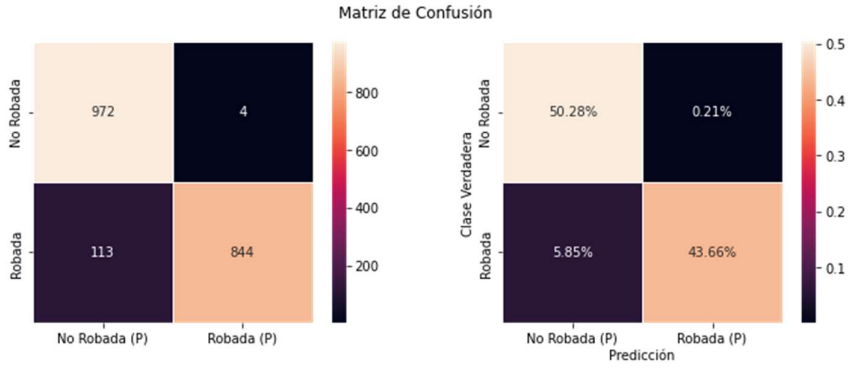

```
# Tomek Links
X_train, X_test, y_train, y_test = train_test_split(x_tl, y_tl,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Árbol de decisión - Tomek Links
Under-Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```



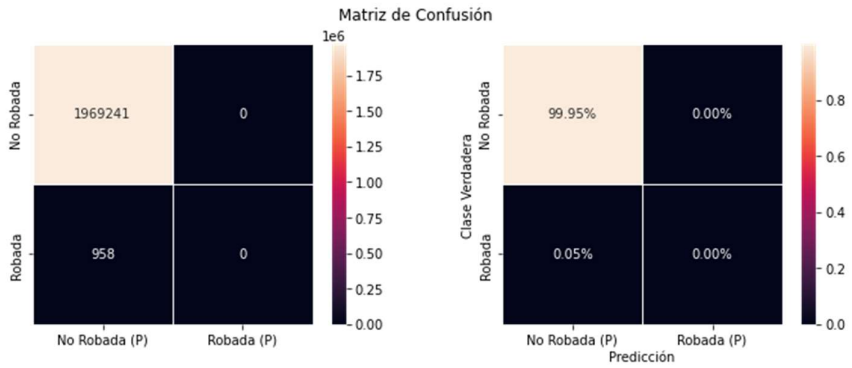
```
# Regresión logística
modelo = LogisticRegression(random_state = seed)


# Random Under Sampling
X_train, X_test, y_train, y_test = train_test_split(x_rus, y_rus,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Regresión Logística - Random Under-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```




```
# Near Miss
X_train, X_test, y_train, y_test = train_test_split(x_nm, y_nm,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Regresión Logística - NearMiss
Under-Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```

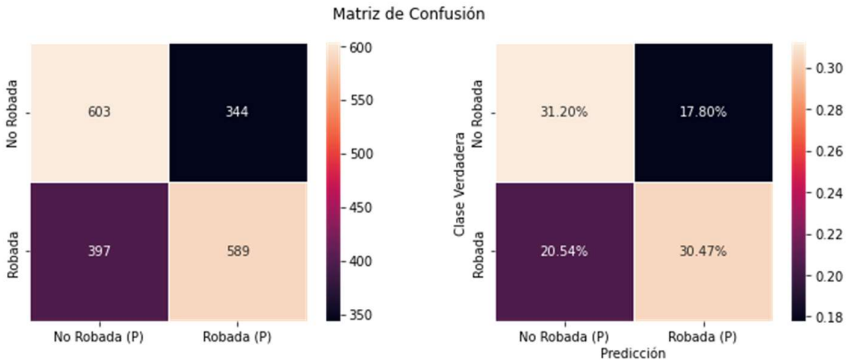



```
# Tomek Links
X_train, X_test, y_train, y_test = train_test_split(x_tl, y_tl,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Regresión Logística - Tomek Links
Under-Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```

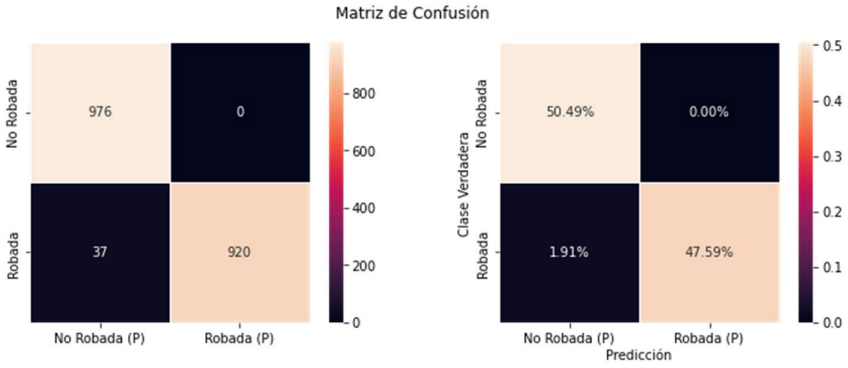



```
# Random Forest
modelo = RandomForestClassifier()


# Random Under Sampling
X_train, X_test, y_train, y_test = train_test_split(x_rus, y_rus,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)
```

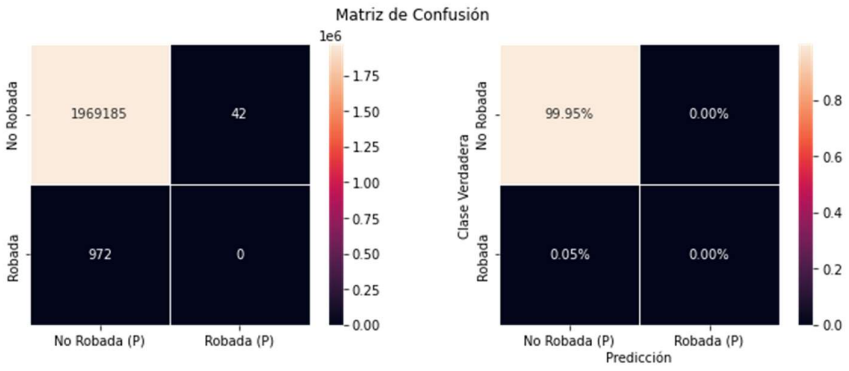
```
resultados = guardo_resultados('Random Forest - Random Under-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```




```
# Near Miss
X_train, X_test, y_train, y_test = train_test_split(x_nm, y_nm,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Random Forest - NearMiss Under-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```

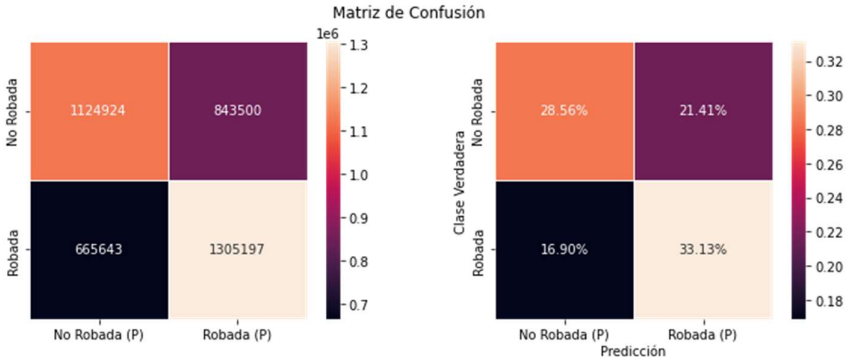


```
# Tomek Links
X_train, X_test, y_train, y_test = train_test_split(x_tl, y_tl,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Random Forest - Tomek Links Under-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```

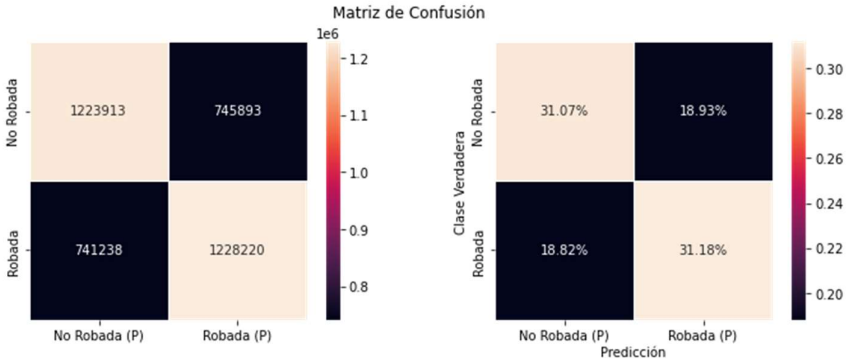



```
# OVER SAMPLING
## Random Over-Sampling
ros = RandomOverSampler(random_state = seed)
x_ros, y_ros = ros.fit_resample(X, y)

## Synthetic Minority Oversampling Technique (SMOTE)
smote = SMOTE(random_state = seed)
x_smote, y_smote = smote.fit_resample(X, y)

## ADASYN Oversampling
adasyn = ADASYN(sampling_strategy="minority",
                random_state = seed)
x_adasyn, y_adasyn = adasyn.fit_resample(X, y)

## Corremos los 3 modelos con los 3 métodos de oversampling
# Árbol de decisión
modelo = DecisionTreeClassifier(criterion = 'entropy', max_depth =
4)


# Random Over Sampling
X_train, X_test, y_train, y_test = train_test_split(x_ros, y_ros,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)
```

```
resultados = guardo_resultados('Árbol de decisión - Random Over-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```

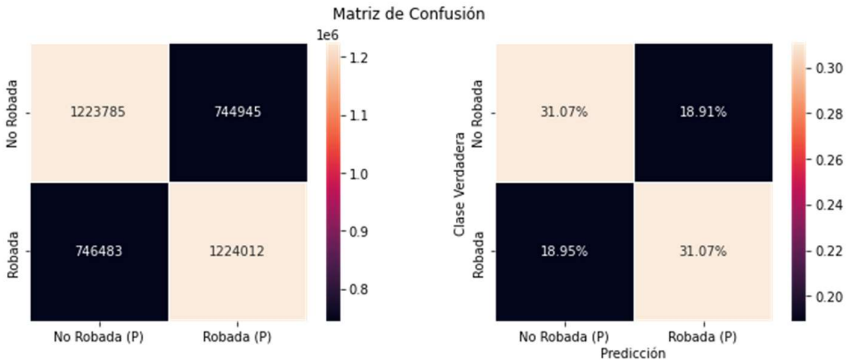



```
# Synthetic Minority Oversampling Technique (SMOTE)
X_train, X_test, y_train, y_test = train_test_split(x_smote,
y_smote, test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Árbol de decisión - SMOTE', y_test,
y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```

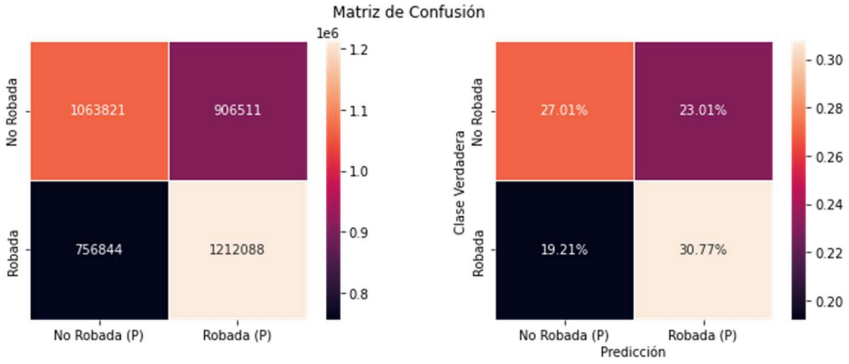



```
# Adasyn Oversampling
X_train, X_test, y_train, y_test = train_test_split(x_adasyn,
y_adasyn, test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)
```

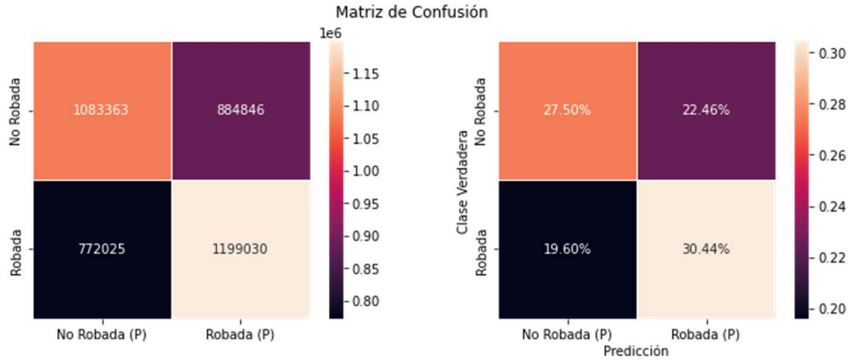
```
resultados = guardo_resultados('Árbol de decisión - ADASYN', y_test,
y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```



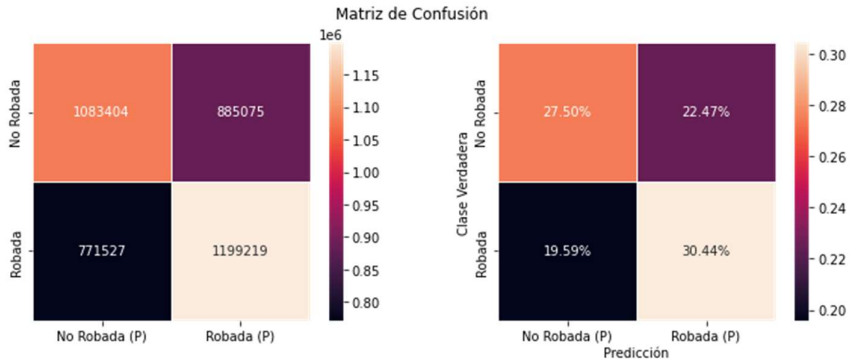

```
# Regresión logística
modelo = LogisticRegression(random_state = seed)

# Random Over Sampling
X_train, X_test, y_train, y_test = train_test_split(x_ros, y_ros,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Regresión Logística - Random Over-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```



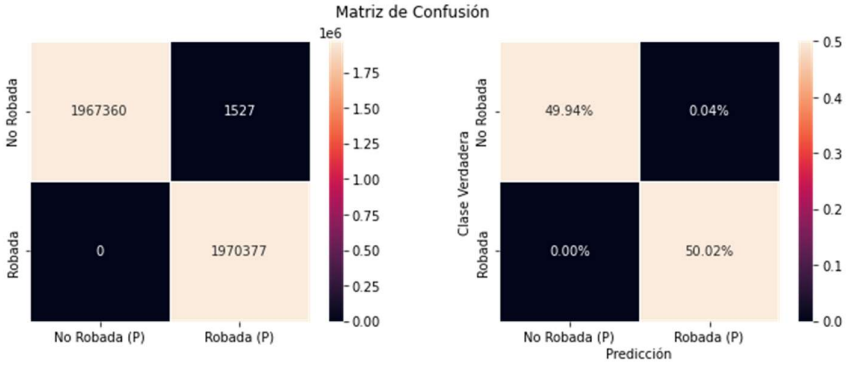
```
# Synthetic Minority Oversampling Technique (SMOTE)
X_train, X_test, y_train, y_test = train_test_split(x_smote,
y_smote, test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Regresión Logística - SMOTE',
y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```



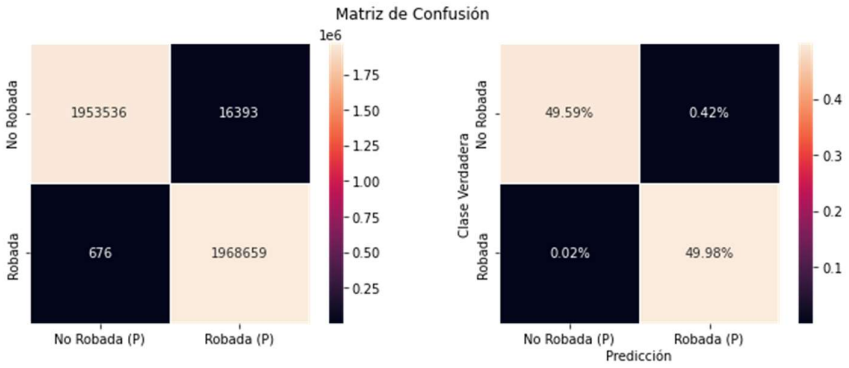

```
# Adasyn Oversampling
X_train, X_test, y_train, y_test = train_test_split(x_adasyn,
y_adasyn, test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Regresión Logística - ADASYN',
y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```



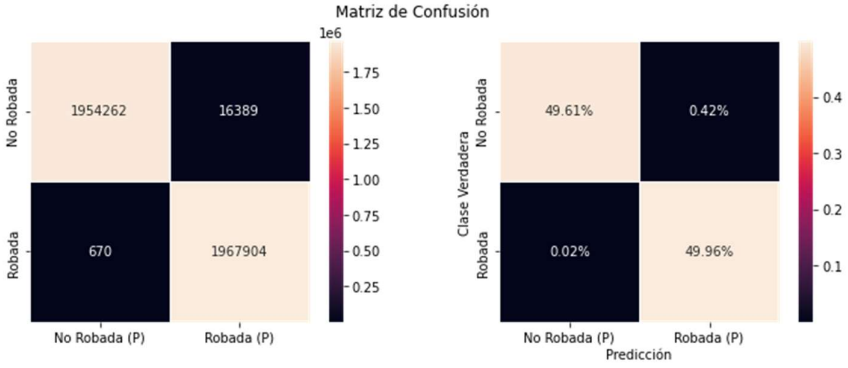
```
# Random Forest
modelo = RandomForestClassifier()

# Random Over Sampling
X_train, X_test, y_train, y_test = train_test_split(x_ros, y_ros,
test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Random Forest - Random Over-
Sampling', y_test, y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```




```
# Synthetic Minority Oversampling Technique (SMOTE)
X_train, X_test, y_train, y_test = train_test_split(x_smote,
y_smote, test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Random Forest - SMOTE', y_test,
y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```



```
# Adasyn Oversampling
X_train, X_test, y_train, y_test = train_test_split(x_adasyn,
y_adasyn, test_size = t_size)
start = time.time()
modelo = modelo.fit(X_train,y_train)
end = time.time()
tiempo_ejecucion = (end - start)
y_pred = modelo.predict(X_test)

resultados = guardo_resultados('Random Forest - ADASYN', y_test,
y_pred, tiempo_ejecucion)
resultados_df = resultados_df.append(resultados, ignore_index=True)
matriz_confusion(y_test, y_pred)
```




```
# Ordenamos los resultados
resultados_df.sort_values(by = ['ROCAUC'], ascending = False,
ignore_index = True, inplace = True)

# Guardo los resultados
def _getToday():
        return dt.date.today().strftime("%Y%m%d")

filename = "%s_%s.%s" % ("/content/gdrive/My Drive/ECD/resultados",
_getToday() ,".csv")

resultados_df.to_csv(filename,
                     index = False,
                     encoding='utf-8')
```