UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

DIAGNÓSTICO DE FALLAS INTELIGENTE EN BOMBAS HIDRÁULICAS CON DISTRIBUCIÓN DESEQUILIBRADA EN LOS ESTADOS DE SALUD

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

RAIMUNDO LORCA OVALLE

PROFESORA GUÍA:
VIVIANA MERUANE NARANJO

MIEMBROS DE LA COMISIÓN:
JORGE MARÍN CASTILLO
BENJAMIN HERRMANN PRIESNITZ

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE: Ingeniero Civil Mecánico
POR: Raimundo Lorca Ovalle
FECHA: 2023
PROF. GUÍA: Viviana Meruane Naranjo

DIAGNÓSTICO DE FALLAS INTELIGENTE EN BOMBAS HIDRÁULICAS CON DISTRIBUCIÓN DESEQUILIBRADA EN LOS ESTADOS DE SALUD

La falla de un equipo en una línea de producción puede generar grandes perdidas económicas, es por esto que es valioso poseer herramientas que permitan monitorear el estado de salud de los equipos, detectando y adelantándose a potenciales fallas. Sin embargo, la escasa cantidad de datos de fallas disponibles en aplicaciones de la industria dificulta el desarrollo de modelos basados en aprendizaje de máquinas, ya que una distribución desequilibrada en los estados de salud (clases) puede afectar significativamente el desempeño de los modelos.

Un enfoque para mitigar el problema de clases desbalanceadas es la implementación de métodos de resampleo, los cuales tienen la función de eliminar o generar instancias con el fin de balancear la cantidad de datos en cada clase. Otro enfoque es la aplicación de métodos de ensamblaje, los cuales combinan clasificadores débiles para obtener un mejor rendimiento.

Con el objetivo de detectar fallas de un equipo bajo la problemática descrita, se estudia, implementa y evalúa la combinación de métodos de resampleo de datos con algoritmos de aprendizaje de máquinas supervisados, tal como redes neuronales y métodos de ensamblaje. Esto aplicado en el contexto del monitoreo de bombas de impulsión de una planta desalinizadora de una minera.

Se obtiene que la combinación de métodos de resampleo y redes neuronales presenta mejores resultados que la combinación de métodos de resampleo con algoritmos de ensamblaje. En donde el modelo con mejor desempeño, Synthetic Minority Over-sampling Technique (SMOTE) + Edited Nearest Neighbours (ENN) + redes neuronales multicapa (RNM), es capaz de detectar la falla con la que se entrena. No obstante, este modelo no es capaz de detectar fallas en un equipo distinto al que fue entrenado, a pesar de que los equipos sean iguales.

Finalmente, se puede afirmar que se logra desarrollar un modelo de diagnóstico de fallas basado en aprendizaje de máquinas que presenta una mejora en el desempeño al trabajar con datos desbalanceados, pero que simultáneamente presenta una capacidad de generalización limitada. Considerando lo anterior, se recomienda entrenar un modelo para cada equipo por separado, ya que de esta manera podría detectar de mejor manera las fallas.

Agradecimientos

Quiero agradecer a todas las personas que de alguna u otra forma me ayudaron durante mi etapa universitaria, tanto académicamente como en el día a día.

A mis padres, por todo lo que me han dado, este momento no sería posible sin ustedes. Me han enseñado lo importante que es el esfuerzo y dedicación en la vida, espero seguir contando con su apoyo.

A mis hermanos, por apoyarme en el día a día, subirme el ánimo y tenerme más fe que nadie.

A la Trini, que ha sido un apoyo irremplazable a lo largo de todo este proceso. Espero que este sea el inicio de un largo camino juntos.

A mis amigos de la universidad, que hicieron que valiera el esfuerzo de seguir cada año, son lo más grande y no podría pedir más. Mención especial al Peter, eterno partner durante todos los años de carrera.

A mis amigos de la vida, que siempre han estado ahí y espero siga así durante mucho tiempo.

Y a la Lova, por ser la acompañante más tierna y fiel durante cada trasnochada trabajando.

Tabla de Contenido

1. Introducción **1**

1.1. Motivación 1
1.2. Contexto 2
1.3. Objetivo general 3
1.4. Objetivos específicos 3
1.5. Alcances 4

2. Antecedentes **5**

2.1. Prepocesamiento 5
2.2. Visualización 5
2.3. Enfoques actuales al problema de datos desbalanceados 6
2.4. Resampleo de datos 7
2.4.1. Oversampling 8
2.4.2. Undersampling 10
2.4.3. Combinación oversampling y undersampling 12
2.5. Ensemble learning 13
2.5.1. Métodos tipo bagging 13
2.5.2. Métodos tipo boosting 15
2.6. Redes neuronales 16
2.7. Métricas de evaluación 19

3. Metodología **21**

3.1. Montaje experimental 21
3.2. Caracterización de datos y falla de entrenamiento 22
3.3. Preprocesamiento 22
3.4. Definición etiquetas y conjuntos 24
3.5. Selección y desarrollo de algoritmos 26
3.6. Visualización gráfica de métodos de resampleo 28
3.7. Selección de hiperparámetros 28
3.8. Evaluación 29

4. Resultados y discusión **30**

4.1. Caracterización de datos y falla de entrenamiento 30

4.2. Preprocesamiento . . . 34
4.3. Visualización gráfica de clases en el caso base . . . 36
4.4. Evaluación del desempeño de los modelos . . . 37
4.4.1. Resumen de los resultados de evaluación . . . 37
4.4.2. Modelo caso base . . . 41
4.4.3. Modelo Random Over Sampler + RNM . . . 42
4.4.4. Modelo SMOTE + RNM . . . 44
4.4.5. Modelo All-KNN + RNM . . . 45
4.4.6. Modelo Neighbourhood Cleaning Rule + RNM . . . 47
4.4.7. Modelo SMOTE-ENN + RNM . . . 48
4.4.8. Modelo Balanced Bagging Classifier . . . 49
4.4.9. Modelo Easy Ensemble Classifier . . . 50
4.5. Evaluación capacidad de generalización . . . 51
4.5.1. Detección de falla de entrenamiento . . . 51
4.5.2. Detección de falla nueva . . . 52

5. Conclusiones y trabajos futuros **55**
5.1. Trabajos futuros . . . 56

Bibliografía **57**

Anexos **61**

Anexo A. Resultados modelos restantes **61**
A.1. Modelo ADASYN + RNM . . . 61
A.2. Modelo Borderline SMOTE + RNM . . . 62
A.3. Modelo SVMSMOTE + RNM . . . 63
A.4. Modelo Cluster Centroids + RNM . . . 64
A.5. Modelo Condensed Nearest Neighbour + RNM . . . 65
A.6. Modelo Edited Nearest Neighbours + RNM . . . 66
A.7. Modelo Repeated Edited Nearest Neighbours + RNM . . . 67
A.8. Modelo Near Miss + RNM . . . 68
A.9. Modelo One Sided Selection + RNM . . . 69
A.10. Modelo Random Under Sampler + RNM . . . 70
A.11. Modelo Tomek Links + RNM . . . 71
A.12. Modelo SMOTE + Tomek Links + RNM . . . 72
A.13. Modelo Balanced Random Forest Classifier . . . 73
A.14. Modelo RUS Boost Classifier . . . 73

Anexo B. Códigos **74**
B.1. Gráficos de dispersión . . . 74
B.2. Definición de conjuntos de entrenamiento y evaluación . . . 77

B.3. Ajuste de hiperparámetros 78
B.4. Entrenamiento y evaluación de modelos 79

Índice de Tablas

2.1. Matriz de confusión 19
4.1. Hiperparámetros entregados por RandomizedSearchCV para el caso base. 42
4.2. Resultados de la evaluación del caso base. 42
4.3. Hiperparámetros entregados por RandomizedSearchCV para modelo Random Over Sampler + RNM. 43
4.4. Resultados de la evaluación del modelo Random Over Sampler + RNM. 44
4.5. Hiperparámetros entregados por RandomizedSearchCV para modelo SMOTE + RNM. 45
4.6. Resultados de la evaluación del modelo SMOTE + RNM. 45
4.7. Hiperparámetros entregados por RandomizedSearchCV para modelo All-KNN + RNM. 46
4.8. Resultados de la evaluación del modelo All-KNN + RNM. 47
4.9. Hiperparámetros entregados por RandomizedSearchCV para modelo Neighbourhood Cleaning Rule + RNM. 48
4.10. Resultados de la evaluación del modelo Neighbourhood Cleaning Rule + RNM. . . . 48
4.11. Hiperparámetros entregados por RandomizedSearchCV para modelo SMOTE-ENN + RNM. 49
4.12. Resultados de la evaluación del modelo SMOTE-ENN + RNM. 49
4.13. Resultados de la evaluación del modelo Balanced Bagging Classifier. 50
4.14. Resultados de la evaluación del modelo Easy Ensemble Classifier. 50
A.1. Hiperparámetros entregados por RandomizedSearchCV para modelo ADASYN + RNM. 61
A.2. Resultados de la evaluación del modelo ADASYN + RNM. 62
A.3. Hiperparámetros entregados por RandomizedSearchCV para modelo BorderlineSMOTE + RNM. 62
A.4. Resultados de la evaluación del modelo BorderlineSMOTE + RNM. 63
A.5. Hiperparámetros entregados por RandomizedSearchCV para modelo SVMSMOTE + RNM. 63
A.6. Resultados de la evaluación del modelo SVMSMOTE + RNM. 63
A.7. Hiperparámetros entregados por RandomizedSearchCV para modelo Cluster Centroids + RNM. 64
A.8. Resultados de la evaluación del modelo Cluster Centroids + RNM. 64
A.9. Hiperparámetros entregados por RandomizedSearchCV para modelo Condensed Nearest Neighbour + RNM. 65
A.10. Resultados de la evaluación del modelo Condensed Nearest Neighbour + RNM. . . . 65

A.11. Hiperparámetros entregados por RandomizedSearchCV para modelo Edited Nearest Neighbours + RNM. 66
A.12. Resultados de la evaluación del modelo Edited Nearest Neighbours + RNM. 66
A.13. Hiperparámetros entregados por RandomizedSearchCV para modelo Repeated Edited Nearest Neighbours + RNM. 67
A.14. Resultados de la evaluación del modelo Repeated Edited Nearest Neighbours + RNM. 67
A.15. Hiperparámetros entregados por RandomizedSearchCV para modelo Near Miss + RNM. 68
A.16. Resultados de la evaluación del modelo Near Miss + RNM. 68
A.17. Hiperparámetros entregados por RandomizedSearchCV para modelo One Sided Selection + RNM. 69
A.18. Resultados de la evaluación del modelo One Sided Selection + RNM. 69
A.19. Hiperparámetros entregados por RandomizedSearchCV para modelo Random Under Sampler + RNM. 70
A.20. Resultados de la evaluación del modelo Random Under Sampler + RNM. 70
A.21. Hiperparámetros entregados por RandomizedSearchCV para modelo Tomek Links + RNM. 71
A.22. Resultados de la evaluación del modelo Tomek Links + RNM. 71
A.23. Hiperparámetros entregados por RandomizedSearchCV para modelo SMOTE-Tomek + RNM. 72
A.24. Resultados de la evaluación del modelo SMOTE-Tomek + RNM. 73
A.25. Resultados de la evaluación del modelo Balanced Random Forest Classifier. 73
A.26. Resultados de la evaluación del modelo RUS Boost Classifier. 73

Índice de Ilustraciones

2.1. Diferentes enfoques al problema de la clasificación de datos desequilibrados [3]. . . 7
2.2. Representación gráfica oversampling. [5] . . . 8
2.3. Representación gráfica undersampling. [5] . . . 10
2.4. Representación gráfica método combinado oversampling y undersampling. [5] . . . 12
2.5. Representación gráfica algoritmo tipo bagging. [21] . . . 14
2.6. Representación gráfica algoritmo tipo boosting. [21] . . . 15
2.7. Representación gráfica red neuronal con una capa oculta. [28] . . . 17
2.8. Representación pesos y sesgo en una red neuronal. [29] . . . 17
3.1. Ejemplo sistema motobomba. . . . 21
3.2. Visualización gráfica de las etiquetas en vibraciones del motor. . . . 25
3.3. Visualización gráfica de las etiquetas en vibraciones del bomba. . . . 25
4.1. Historial de monitores de las 3 fases de corriente del motor. . . . 30
4.2. Historial de monitoreo del caudal entregado por el sistema. . . . 31
4.3. Historial de monitoreo de la temperatura en 4 puntos. . . . 32
4.4. Historial de monitoreo del nivel de vibraciones del motor. . . . 33
4.5. Historial de monitoreo del nivel de vibraciones de la bomba. . . . 33
4.6. Temperaturas después del preprocesamiento. . . . 35
4.7. Nivel vibraciones bomba después del preprocesamiento. . . . 35
4.8. Nivel vibraciones motor después del preprocesamiento. . . . 36
4.9. Visualización datos originales en dos dimensiones. . . . 37
4.10. Resumen f1-score testeo modelos. . . . 38
4.11. Resumen desviación estándar f1-score testeo modelos. . . . 40
4.12. Resultados resampleo Random Over Sampler en dos dimensiones. . . . 43
4.13. Resultados resampleo SMOTE en dos dimensiones. . . . 44
4.14. Resultados resampleo All-KNN en dos dimensiones. . . . 46
4.15. Resultados resampleo Neighbourhood Cleaning Rule en dos dimensiones. . . . 47
4.16. Resultados resampleo SMOTE-ENN en dos dimensiones. . . . 48
4.17. Visualización detección falla de entrenamiento. . . . 52
4.18. Visualización detección falla de entrenamiento. . . . 53
A.1. Resultados resampleo ADASYN en dos dimensiones. . . . 61
A.2. Resultados resampleo Borderline SMOTE en dos dimensiones. . . . 62
A.3. Resultados resampleo SVMSMOTE en dos dimensiones. . . . 63
A.4. Resultados resampleo Cluster Centroids en dos dimensiones. . . . 64

A.5. Resultados resampleo Condensed Nearest Neighbour en dos dimensiones. 65
A.6. Resultados resampleo Edited Nearest Neighbours en dos dimensiones. 66
A.7. Resultados resampleo Repeated Edited Nearest Neighbours en dos dimensiones. . . 67
A.8. Resultados resampleo Near Miss en dos dimensiones. 68
A.9. Resultados resampleo One Sided Selection en dos dimensiones. 69
A.10. Resultados resampleo Random Under Sampler en dos dimensiones. 70
A.11. Resultados resampleo Tomek Links en dos dimensiones. 71
A.12. Resultados resampleo SMOTE-Tomek Links en dos dimensiones. 72

Capítulo 1

Introducción

1.1. Motivación

Si una máquina importante en una línea de producción falla súbitamente, las repercusiones económicas probablemente serán considerables. Por lo que se vuelve beneficioso tener herramientas de mantenimiento eficientes que permitan evitar fallas antes de que sucedan.

Es debido a esto que determinar el momento óptimo para hacer mantención a un equipo ha sido un tema de estudio importante durante el último tiempo, en donde se ha intentado desarrollar modelos que automaticen el proceso sin perder eficacia. Una de las ramas claves que estudia esta problemática es el diagnóstico de fallas inteligente o "Intelligent fault diagnosis", IFD de ahora en adelante, la cual estudia la aplicación de métodos de aprendizaje de máquinas en el diagnóstico de fallas en equipos.

La implementación de estos métodos presenta la gran ventaja de no depender del juicio humano de individuos especializados en el área, ya que los algoritmos en cuestión, están automatizados. Esto permite ahorrar valiosos recursos, tales como tiempo y dinero, además de mejorar la gestión de los activos físicos. Es debido a esto que IFD ha atraído la atención de muchos investigadores e ingenieros, tendencia que está profundamente relacionada con el desarrollo de nuevas técnicas de aprendizaje de máquinas. Lo que se ha traducido en un considerable aumento en el número de publicaciones relacionadas a IFD durante la última década [1].

Si bien hay numerosos estudios que han explorado esta área, la gran mayoría de los modelos propuestos han sido desarrollados y validados con datos obtenidos en ambientes académicos, mientras que todavía son escasas las investigaciones que abordan este problema en aplicaciones reales de la industria.

La gran diferencia entre ambos casos es la calidad de los datos con los que se trabaja para desarrollar los modelos. En el ambiente académico se generan los datos experimentalmente, obteniendo así datos ideales, los cuales están limpios y balanceados. Por otra parte, en los casos ingenieriles

reales, se tiene que los datos obtenidos distan de ser óptimos para el desarrollo de este tipo de modelos.

Uno de los problemas más comunes de este tipo de base de datos consiste en que se tiene una distribución desequilibrada en los estados de salud, ya que es común que la mayoría de los datos sean recopilados cuando el equipo esté en un estado saludable. En otras palabras, hay muy poca información sobre el equipo cuando este presenta fallas, por lo que se tiene escaso conocimiento de cómo se comportan los datos en estos casos.

El nivel de desbalance de una base de datos se puede cuantificar como la razón entre la cantidad de datos pertenecientes a la clase minoritaria y la cantidad de datos de la clase mayoritaria. En el contexto de los estados de salud de un sistema mecánico, la clase minoritaria corresponde al conjunto de datos catalogados como falla y la clase mayoritaria a los datos catalogados como saludable.

En el marco de modelos de aprendizaje de máquinas, si se utilizan datos desequilibrados para entrenar los modelos de diagnóstico, el límite de decisión de estos podría verse forzado a cambiar hacia los estados de salud con una mayoría de instancias, lo que podría generar un desempeño deficiente del modelo entrenado. Teniendo esto en cuenta, se tiene la motivación de abordar este desafío con datos reales de la industria.

1.2. Contexto

Una alternativa comúnmente utilizada para suplir la demanda hídrica de las mineras es la implementación de plantas desalinizadoras. En este caso de estudio se utilizan datos de una planta desalinizadora del tipo osmosis inversa, la cual tiene una capacidad de tratamiento del orden de 2.000 [l/s]. Cabe mencionar que no se puede exponer detalles de la planta debido a un acuerdo de confidencialidad con la empresa que provee los datos.

Las principales áreas operacionales que conforman la planta desalinizadora son:

1. Sistema de captación de agua marina.
2. Sistema de pretratamiento y de ósmosis inversa.
3. Sistema de estaciones de bombeo para transportar agua procesada.
4. Reservorios y distribución del agua procesada.

Con respecto al sistema de estaciones de bombeo, este consiste en múltiples estaciones que transportan el agua procesada, las cuales contienen un total de 60 bombas de impulsión aproximadamente. En cada estación de bombeo se distribuyen de manera paralela una serie de bombas de impulsión, las cuales alimentan el reservorio de la estación siguiente.

Cada bomba de impulsión consiste en un sistema compuesto por un motor y una bomba centrifuga de 5 etapas, en donde cada motor trabaja a 2986 [RPM] y tiene una potencia de 3500 [kW], entregando un caudal aproximado de 1000 [m^3/h]. En adición, cada sistema motor-bomba está equipado con un conjunto de sensores que recopilan información sobre el estado de salud. A continuación se listan las variables físicas medidas:

- 8 sensores de vibración.
- 4 sensores de temperatura.
- 3 sensores de corriente del motor.
- 1 sensor de caudal de la bomba.

En este caso de estudio se trabaja con el historial de monitoreo de varios sistemas motor-bomba de las estaciones de bombeo. A pesar de la gran disponibilidad de datos, solo algunos historiales de monitoreo sirven para poder entrenar un modelo de aprendizaje de máquinas, esto debido a la calidad de los datos.

Idealmente, para el desarrollo de un problema de aprendizaje de máquinas, se espera tener una razón de desbalance cercana a 1:1, es decir, cada un dato de la clase mayoritaria hay uno de la minoritaria. Mientras que para este caso de estudio se tiene una razón de 1:49, es decir, cada 49 datos saludables solo hay 1 que representa una falla. Por lo que la base de datos está sumamente desbalanceada, dificultando el desarrollo de un algoritmo capaz de detectar fallas eficientemente.

1.3. Objetivo general

Diseñar un algoritmo de diagnóstico de fallas basado en métodos de aprendizaje de máquinas, tal que sea capaz de presentar un buen desempeño al trabajar con un conjunto de datos con estados de salud desequilibrados.

1.4. Objetivos específicos

- Buscar bibliografía sobre diferentes algoritmos de aprendizaje de máquinas y metodologías que tengan un buen desempeño con datos desbalanceados.
- Preprocesar datos de historiales de monitoreo de bombas hidráulicas para facilitar la toma de decisiones del modelo a desarrollar.
- Evaluar el desempeño de un conjunto de técnicas seleccionadas para la detección de fallas en bombas de impulsión.
- Seleccionar modelo que entregue los mejores resultados y evaluar su capacidad de generalización.

1.5. Alcances

Este estudio se limita a desarrollar un algoritmo enfocado a detectar fallas de una bomba hidráulica en particular, implicando que tenga una capacidad limitada de generalización a otras aplicaciones.

Capítulo 2

Antecedentes

2.1. Prepocesamiento

Antes de iniciar cualquier proyecto de manejo de datos, es imperativo hacer un análisis manual sobre los datos que se tienen. La importancia de esto se debe a que la gran mayoría de los casos, los datos vienen con una baja calidad, presentando problemas tales como, formato incompatible, datos duplicados, datos faltantes, datos irrelevantes, entre otros. Este análisis previo permite identificar todos estos problemas, además de entregar una noción más clara de cómo se compone. Teniendo esto en mente, se vuelve claro que es necesario limpiar los datos antes de intentar encontrar patrones y extraer conocimiento de estos.

Una vez identificado los problemas más críticos de los datos, se debe proceder a aplicar los métodos de limpieza de datos pertinentes al caso. Para este paso, hay numerosas opciones, destacando la eliminación de datos anómalos que representan ruido, los cuales ensucian los datos, en específico, datos correspondientes a las detenciones de un equipo, ya que estos no aportan información valiosa, sino todo lo contrario.

Luego, para el caso de los datos recolectados por sensores de proximidad (datos de vibraciones), es beneficioso aplicar un preprocesamiento adicional, ya que se busca medir las vibraciones con un sistema de referencia en el centro del eje. Para lograr esto, se aplica el método de la media móvil, el cual consiste en el cálculo de la media de una ventana de datos y su resta para cada dato dentro de esa ventana temporal.

2.2. Visualización

Al trabajar con bases de datos con numerosas variables, la visualización de los datos se torna complicada, y aún más la inferencia a partir de los resultados. Una forma de abordar este problema es aplicar métodos de reducción de dimensionalidad, los cuales tienen diversos beneficios, tal como poder reducir el espacio a 2 o 3 dimensiones, permitiendo visualizar de manera grafica todos los resultados y poder encontrar patrones o comportamientos de manera más fácil. Además de esto,

el espacio necesario para almacenar todos los datos disminuye considerablemente, lo que baja el costo computacional del modelo.

Un ejemplo de estos métodos es el análisis de componentes principales (PCA) [2], el cual busca la proyección espacial o hiperplano, tal que un conjunto de variables correlacionadas se convierta en un conjunto de variables no correlacionadas linealmente. Esta proyección tiene el nombre de componentes principales y busca maximizar la varianza entre las variables proyectadas.

Este tipo de métodos suelen ser efectivos si se aplican de manera correcta y pertinente, pero al transformar los datos originales, se pierde interpretabilidad del problema, ya que las variables originales, en este caso variables físicas, se pierden. En específico, para el método de análisis de componentes principales, las variables se convierten en componentes que no tienen una interpretación física directa, por lo que no se sabe qué variables son las que inciden de mayor manera en la toma de decisiones posterior al preprocesamiento. Teniendo en cuenta lo anterior, se debe tener precaución al interpretar las representaciones visuales obtenidas.

2.3. Enfoques actuales al problema de datos desbalanceados

Una de las mejores herramientas computacionales para detectar las fallas en un equipo son los clasificadores, sin embargo, cuando se presenta el problema de distribución de clases desequilibrada en un conjunto de datos, estas herramientas han encontrado una seria dificultad para llevar a cabo su cometido [3]. Ya que estos tipos de algoritmos de aprendizaje de máquinas asumen una distribución relativamente balanceada para desempeñarse de buena manera, implicando que cuando se presenta el caso contrario, podrían clasificar todos los datos como parte de la clase mayoritaria y aun así obtener buenos resultados bajo métricas de evaluación estándares.

La problemática se encuentra en que la clase minoritaria suele ser la de interés, por ejemplo, al momento diagnosticar una enfermedad que podría ser letal, tiene mayores consecuencias clasificar a alguien como sano cuando realmente está enfermo, por lo que interesa tener un modelo capaz de detectar eficientemente cuando el paciente está enfermo (clase minoritaria). A continuación, se verán los principales enfoques para abordar el problema de clasificación con datos desbalanceados, los cuales se pueden resumir en la figura 2.1.

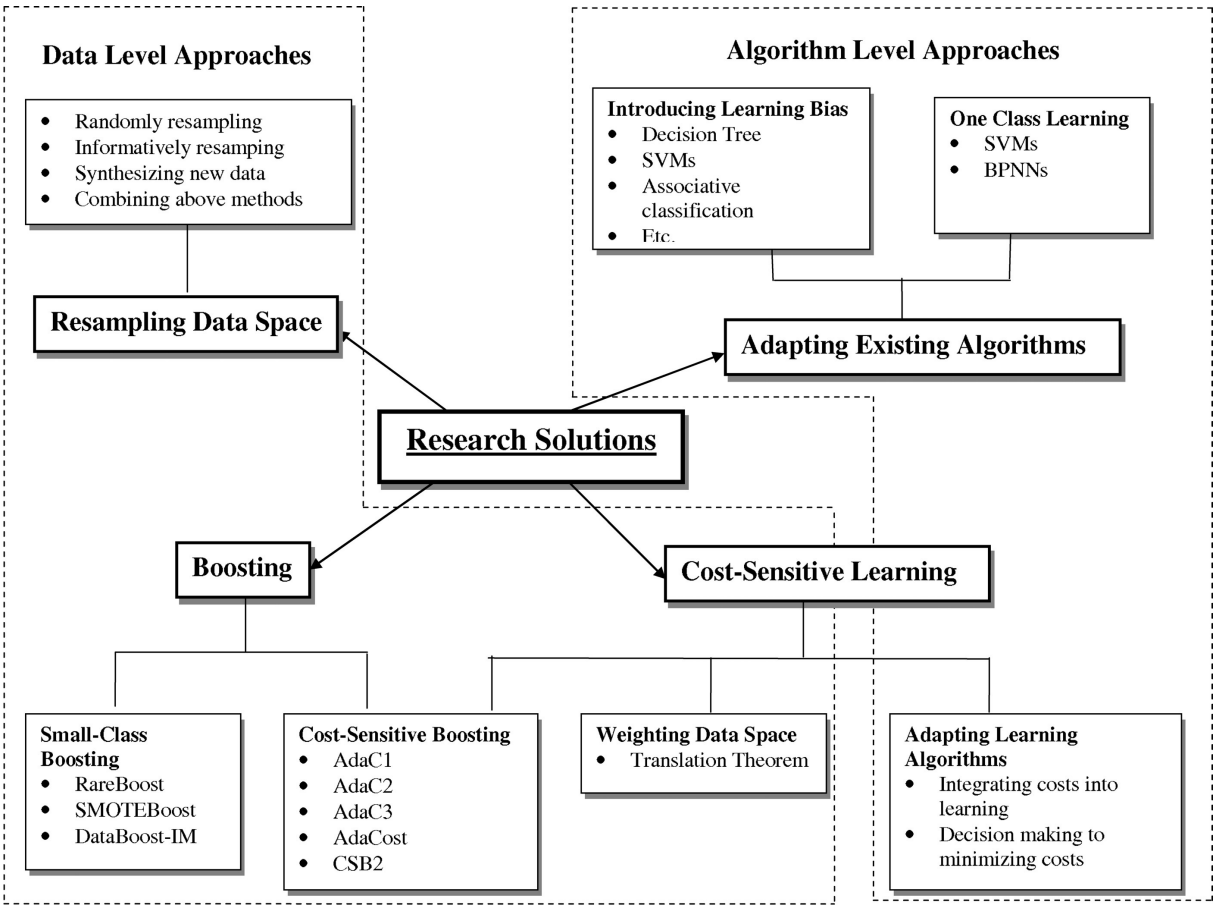

Figura 2.1: Diferentes enfoques al problema de la clasificación de datos desequilibrados [3].

2.4. Resampleo de datos

Con respecto al enfoque de resampleo de datos o "Data level approach", estos se concentran en modificar el conjunto de datos de entrenamiento para disminuir el nivel de desbalance, con el fin de hacerlo más adecuado para los algoritmos de aprendizaje de máquinas. Dentro de este enfoque, se puede distinguir soluciones que generan nuevos datos para las clases minoritarias (oversampling o sobremuestreo), otras que eliminan datos de las clases mayoritarias (undersampling o submuestreo), y por último, aquellas que combinan ambas soluciones mencionadas.

En general, los métodos estándares de resampleo de datos utilizan elemento aleatorios para generar o eliminar datos, lo que puede llevar a eliminar datos de gran importancia o generar datos que no aportan nada. Debido a esto, se vio la necesidad de desarrollar métodos más avanzados, los cuales proponen respetar las distribuciones subyacentes presentes y eliminar datos anómalos que podrían perjudicar negativamente los algoritmos de aprendizaje de máquinas [4].

A continuación se detallan cada uno de los enfoques de resampleo de datos y sus métodos más

destacables.

2.4.1. Oversampling

Como se mencionó anteriormente, el oversampling consiste en generar datos nuevos de la clase minoritaria para suavizar el efecto de desbalance entre clases. En la figura 2.2, se puede ver como se toma la base de datos original y se balancean las clases al generar datos nuevos en la clase minoritaria.

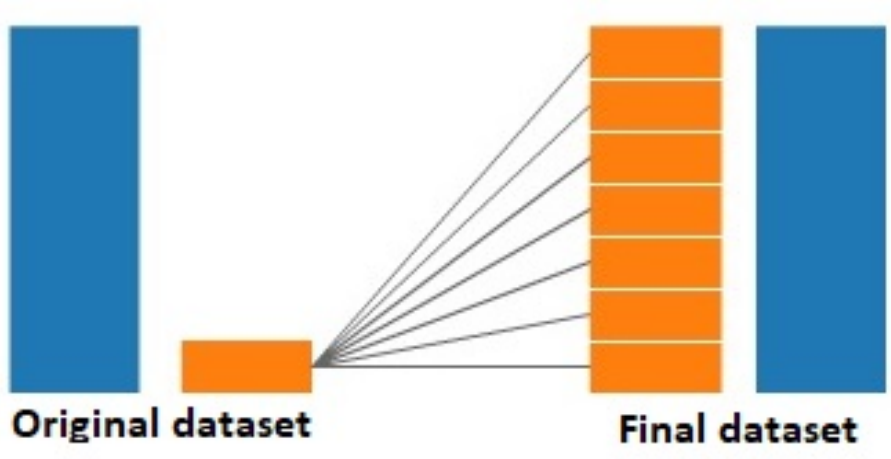

Figura 2.2: Representación gráfica oversampling. [5]

La acción de generar datos nuevos se puede realizar de muchas maneras, tanto de manera aleatoria al duplicar datos ya existentes, como de manera sintética al utilizar técnicas que mejoran la razón de desbalance. Una de las ventajas más grande de este tipo de métodos es que no hay una perdida de datos durante el proceso de resampleo, ya que no se están eliminando datos, como en el caso de los métodos de undersampling. Sin embargo, los métodos de oversampling presentan la desventaja de que al aumentar la cantidad de datos, estos se vuelven más costosos computacionalmente, e incluso, muchos de los datos nuevos podrían ser datos anómalos, los cuales no aportan al desarrollo del modelo y pueden considerar como ruido.

Otra desventaja es que existe una tendencia a que el algoritmo de aprendizaje de máquinas sufra de overfitting o sobreajuste, es decir, el criterio de decisión se ve limitado por los datos de entrenamiento y tiene una mala capacidad de clasificación con datos nuevos.

Sin duda estos métodos han sido de interés a lo largo de los años, es por esto que hay un gran número de algoritmos que aprovechan el concepto de oversampling para combatir el problema de clases desbalanceadas. A continuación se muestran algunos de los algoritmos de oversampling más importantes:

Random Over Sampling (ROS): En este método, el desbalance entre clases es combatido al duplicar aleatoriamente instancias de la clase minoritaria [6]. La gran desventaja de este método

es que puede aumentar la probabilidad de sufrir overfitting, ya que el método consiste en duplicar información.

Adaptive Synthetic Sampling (ADASYN): La idea principal de este método es utilizar una distribución ponderada para diferentes datos pertenecientes a la clase minoritaria, esto según su nivel de dificultad en el aprendizaje. En donde se generan más datos sintéticos para ejemplos de la clase minoritaria que son más difíciles de aprender en comparación con los otros. Para lograr esto, utiliza el método "K-nearest neighbor" (KNN) para seleccionar las regiones con la densidad deseada y generar datos sintéticos dentro de esta [7].

Synthetic Minority Over-sampling Technique (SMOTE): En este método, el sobremuestreo es realizado al seleccionar datos de la clase minoritaria y aplicando interpolación entre ellos, es decir, se define una recta entre dos datos y se genera una nueva muestra en un punto a lo largo de ella. De forma más específica, se elige una muestra de la clase minorita, luego con el método KNN se definen los k vecinos más cercanos, se elige uno entre ellos y se realiza la interpolación entre ambos puntos seleccionados [8].

Una de las principales debilidades de SMOTE es que puede generar overfitting, ya que este método generaliza ciegamente datos en la clase minoritaria sin verificar el estado de las instancias como buenas o ruido. Esto puede suceder si es que las clases no están bien definidas y hay un grado de superposición entre ellas, generando que la interpolación puede expandir la clase minoritaria y aumentar el grado de superposición entre las clases [9].

Borderline SMOTE (B-SMOTE): Con el fin de mejorar el rendimiento de SMOTE, se propone una modificación de este, en donde solo se generan instancias nuevas cercanas al límite entre la clase minoritaria y mayoritaria. Esto debido a que los datos de esa región suelen ser clasificados erróneamente, por lo que se vuelven más importante durante el proceso de clasificación [9]. Este método utiliza el método de los vecinos más cercanos (K nearest neighbors, o KNN según sus siglas) para identificar aquellos datos que al menos la mitad de sus vecinos son de la misma clase, luego los emplea para generar instancias nuevas [10].

Support Vector Machine SMOTE (SVM-SMOTE): Al igual que el método detallado anteriormente, SVM-SMOTE genera datos sintéticos en la región cercana al límite de decisión, ya que enfocarse en las instancias en esta región puede ayudar a mejorar el desempeño. En específico, este método aproxima el área cercana al límite de decisión mediante vectores de apoyo obtenidos a través del entrenamiento de clasificadores del tipo "Support Vector Machine" (SVM). Las instancias nuevas se generan aleatoriamente a lo largo de las rectas que unen cada vector de apoyo de la clase minoritaria con sus vecinos más cercanos, utilizando técnicas de interpolación y extrapolación, en donde la elección de esta depende de la densidad de instancias de la clase mayoritaria alrededor del dato de interés [11].

2.4.2. Undersampling

Por otro lado, el undersampling consiste en la eliminación de datos pertenecientes a la clase mayoritaria, lo que se puede lograr tanto con métodos aleatorios como sintéticos, al igual que en el caso de oversampling. En la figura 2.3 se puede observar una representación gráfica del funcionamiento de este tipo de métodos, los cuales toman la base de datos original y eliminan datos de la clase mayoritaria, balanceado la cantidad de datos en cada clase.

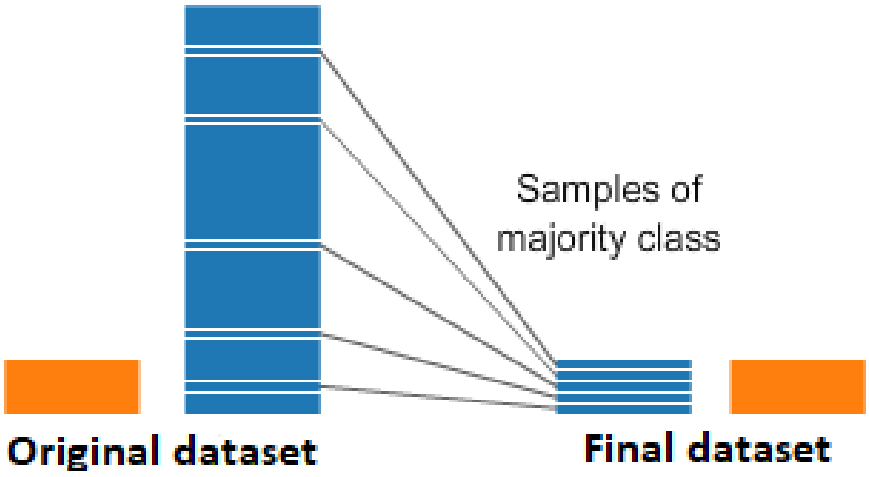

Figura 2.3: Representación gráfica undersampling. [5]

Una de las grandes ventajas que tiene en comparación con los otros métodos de resampleo de datos es que es más barato computacionalmente, ya que se trabaja con una menor cantidad de datos. Sin embargo, estos métodos presentan una gran debilidad, la cual consiste en que se puede perder información importante durante el proceso de resampleo, lo que perjudicaría severamente el desempeño general. A continuación se muestran algunos de los algoritmos de undersampling más importantes:

Random Under Sampler (RUS): Este método consiste en la técnica de undersampling más simple, la cual aborda el problema de desbalance al seleccionar aleatoriamente instancias de la clase mayoritaria y eliminarlas. A pesar de no eliminar las instancias a través de ningún criterio inteligente, tiene la ventaja de que tiene un bajo costo computacional, lo que hace compatible su combinación con métodos de ensamblaje [9].

Tomek Links: Método en donde se detectan los llamados "Tomek's links", los cuales existen cuando dos muestras de diferentes clases son los vecinos más cercanos entre sí. En otras palabras, un Tomek's link entre dos muestras de diferentes clases x e y se define tal que para cada muestra z:

$$d(x,y) < d(x,z) \wedge d(x,y) < d(z,y) \tag{2.1}$$

En donde $d()$ consiste en la distancia entre dos muestras. Una vez que se detecta el Tomek's

link, el algoritmo procede a eliminar la muestra correspondiente a la clase mayoritaria, o si es que se desea, ambas muestras [12].

Edited Nearest Neighbours (ENN): Este método toma el algoritmo de los vecinos más cercanos y edita la base de datos al eliminar muestras que no son parecidas a sus vecinos. Esto se logra al encontrar los K vecinos más cercanos para cada muestra, luego se verifica si la clase mayoritaria del vecindario es igual a la clase de la muestra en cuestión, si estas son diferentes, entonces la muestra y sus K vecinos más cercanos son eliminados de la base de datos (generalmente K=3) [13].

Repeated Edited Nearest Neighbours: Consiste en repetir múltiples veces el método ENN, eliminando así una mayor cantidad de muestras [14].

All-KNN: Consiste en repetir múltiples veces el método ENN e ir aumentando el número de vecinos más cercanos (K) en cada iteración [14]. Una de las ventajas de los últimos tres métodos abordados es que limpian la base de datos al eliminar muestras que están cerca al límite de decisión, lo que facilita el posterior trabajo del clasificador.

Condensed Nearest Neighbour (CNN): El objetivo de este método es eliminar muestras que pertenecen a la clase mayoritaria de tal manera que no haya una perdida de información importante y que las clases se balanceen. Para lograr esto, CNN hace uso del algoritmo de K vecinos más cercanos (KNN), para decidir iterativamente si cada muestra debería ser eliminada o no [15]. En específico, se logra mediante los siguientes pasos [16]:

1. Asignar todas las muestras de la clase minoritaria en un conjunto C.
2. Agregar una muestra de la clase mayoritaria en el conjunto C, todas las otras muestras se asignan a un conjunto S.
3. Se chequea si cada muestra del conjunto S es clasificada correctamente por un clasificador KNN, con K=1.
4. Si es clasificada erróneamente, se asigna al conjunto C, en el caso contrario, no se realiza nada.
5. Repetir proceso en el conjunto S, hasta que no haya ninguna muestra que agregar a C.

Neighborhood Cleaning Rule: Este método consta de dos etapas, primero aplica ENN, eliminando todas las muestras ambiguas de la clase minoritaria, es decir, aquellos datos que se encuentran cerca del límite de decisión. Luego, se aplica CNN, el cual se encarga de eliminar las muestras redundantes que se encuentran lejos del límite de decisión [17].

One Sided Selection: En este método, se emplea Tomek Links para eliminar las muestras cercanas al límite de decisión y luego se utiliza CNN, el cual elimina las muestras lejanas al límite de

decisión. Con la particularidad de que CNN solo realiza una iteración, es decir, no se lleva a cabo el último paso detallado de CNN [18].

Cluster Centroids: Este método submuestrea la clase mayoritaria al reemplazar un clúster de la misma clase, por los centroides de los clústeres entregados por el algoritmo de agrupamiento Kmeans. En otras palabras, Cluster Centroids mantiene N muestras de la clase mayoritaria al ajustar Kmeans con N clústeres y utilizando las coordenadas de los centroides de cada clúster como las nuevas muestras [16]. Cabe mencionar que un clúster es una colección de muestras que poseen características similares entre sí y son diferentes de las muestras de clústeres.

Near Miss: Este método se basa en la técnica de los vecinos más cercanos (KNN) y tiene tres variaciones, en donde cada una emplea diferentes criterios para eliminar muestras de la clase mayoritaria [19]. A continuación se detalla brevemente cada una de estas:

- **Near Miss 1:** Selecciona las muestras de la clase mayoritaria que tienen la menor distancia promedio a las tres muestras más cercanas de la clase minoritaria.
- **Near Miss 2:** Selecciona las muestras de la clase mayoritaria que tienen la menor distancia promedio a las tres muestras más alejadas de la clase minoritaria.
- **Near Miss 3:** Selecciona un número determinado de muestras de la clase mayoritaria por cada una de las muestras más cercanas de la clase minoritaria.

2.4.3. Combinación oversampling y undersampling

Por último, con el fin de aprovechar las ventajas y mitigar las desventajas de cada uno de los tipos de métodos de resampleo, se desarrollan métodos que combinan ambos, es decir, utilizan tanto oversampling como undersampling para balancear las clases de la base de datos. En la figura 2.4 se puede ver como primero se aplica un método de oversampling y luego un método de undersampling, resultado en una base de datos con clases balanceadas.

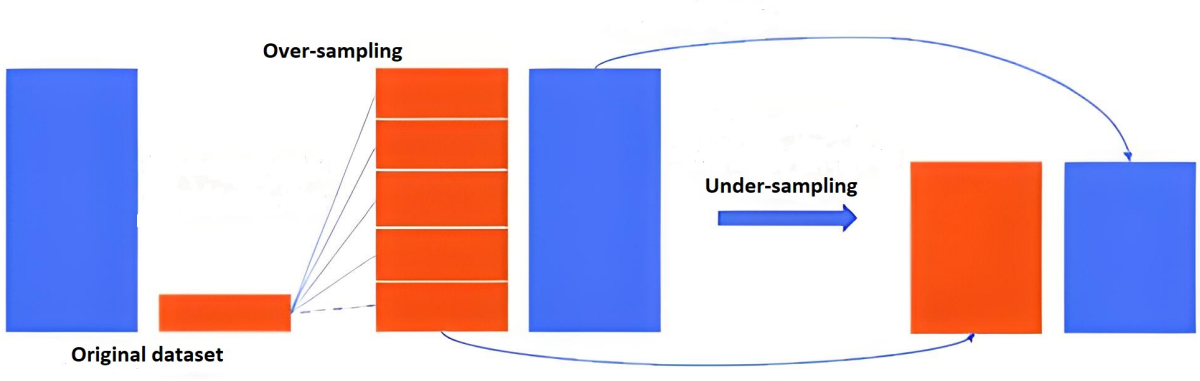

Figura 2.4: Representación gráfica método combinado oversampling y undersampling. [5]

A continuación se muestran algunos de los algoritmos más importantes que combinan oversampling y undersampling:

SMOTE + Tomek links: Con el fin de crear clases mejor definidas en el espacio (sin superposición) y evitar un posterior sobreajuste en el clasificador, se desarrolla un algoritmo que aplica el método de undersampling Tomek links al conjunto de datos previamente sobremuestrado por el método SMOTE. Esto con la particularidad de que el método Tomek links no elimina solo el dato que pertenece a la clase mayoritaria, sino que elimina ambos, lo que permite obtener clases mejor definidas [6].

SMOTE + ENN: Al igual que el método anterior, SMOTE + Edited Nearest Neighbours (ENN) se desarrolla con el objetivo de definir mejor la separación entre la clase mayoritaria y minoritaria. En general, ENN tiende a eliminar más instancias que Tomek links, por lo que este método entrega una limpieza más profunda. En parte se debe a que en este caso ENN elimina instancias de ambas clases, igual al caso anterior, pero además porque cualquier muestra mal clasificada por sus tres vecinos más cercanos se elimina del conjunto de datos [6].

2.5. Ensemble learning

La premisa de los clasificadores tipo ensemble learning es combinar múltiples clasificadores débiles, es decir, clasificadores con una capacidad de predicción reducida, y así construir un clasificador con buen desempeño al combinar la predicción de cada uno mediante una votación. La principal motivación de realizar esto es mejorar la capacidad de generalización del clasificador, en otras palabras, de tener un buen desempeño al clasificar datos nuevos. Debido a que cada clasificador es entrenado con una cantidad limitada de datos, estos suelen cometer fallas, pero los patrones que son clasificadores erróneamente no necesariamente son los mismos, lo que puede mejorar la generalización del algoritmo [3].

El enfoque de ensemble learning es uno los más populares para combatir el problema de clases desequilibradas [20], pero esto no se debe a su naturaleza intrínseca, sino a que se combina el enfoque de ensemble learning con otras técnicas, tales como el resampleo de datos, lo cual mejora el desempeño. Estas técnicas híbridas no modifican el algoritmo del clasificador en sí, sino que preprocesan los datos para que las técnicas de ensemble learning funcionen de mejor manera con datos desbalanceados.

2.5.1. Métodos tipo bagging

Los métodos de ensemble learning tipo bagging consiste en la combinación de clasificadores débiles de forma paralela, en donde cada uno es entrenado con un subconjunto de datos de la base de datos original. Lo que presenta la ventaja de que cada clasificador es entrenado por un subconjunto de datos independiente, por lo que cada clasificador aprende diferentes patrones, permitiendo así

aumentar la capacidad de generalización del clasificador final.

Aunque este tipo de métodos mejore ciertos aspectos que suelen afectar los clasificadores regulares, también se ve afectado por el problema de clases desbalanceadas, ya que este se transmite hacia los subconjuntos que se utilizan para entrenar cada clasificador débil. Debido a esto, se suele aplicar métodos de resampleo de datos en los subconjuntos, para así disminuir el nivel de desbalance de estos.

En la figura 2.5 se puede ver gráficamente el proceso de construcción de un algoritmo tipo bagging.

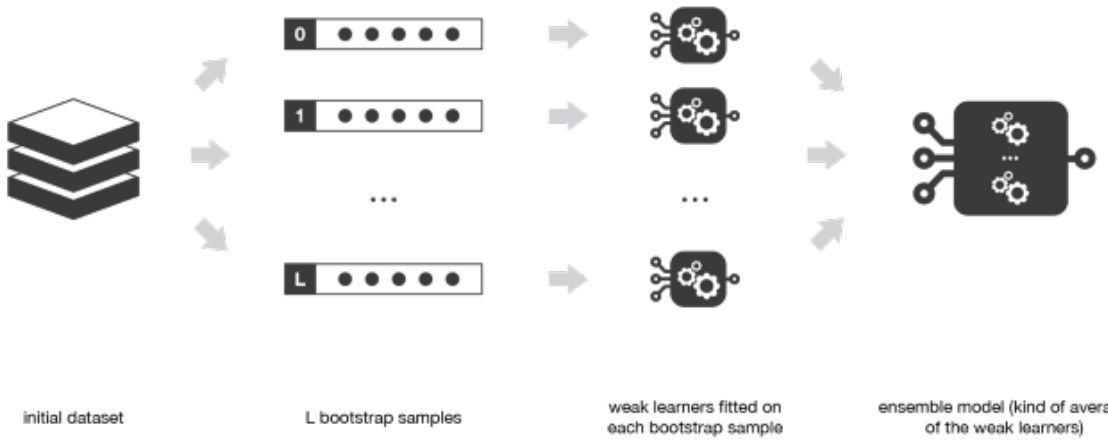

Figura 2.5: Representación gráfica algoritmo tipo bagging. [21]

A continuación se muestran algunos de los algoritmos de ensemble learning tipo bagging más importantes:

Balanced Bagging Classifier: Este clasificador consiste en un conjunto de árboles de decisión, con la particularidad de que cada subconjunto de datos con el cual se entrena cada clasificador débil es resampleado previamente. Para lograr esto, se puede utilizar diversos métodos de resampleo, por lo que existen diferentes variaciones, tal como "Exactly Balanced Bagging" [22], el cual utiliza el método random under sampler, también existe el método "Over-Bagging" [22], el cual utiliza el random over sample, y por último, el método "SMOTE-Bagging", el cual aplica SMOTE [23].

Balanced Random Forest Classifier: Este clasificador consiste en un bosque aleatorio (random forest), en donde cada subconjunto es balanceado mediante el método de resampleo random under sampler [24].

2.5.2. Métodos tipo boosting

Por otro lado, para el caso de los métodos de ensemble learning tipo boosting, estos también consisten en una combinación de clasificadores débiles, pero estos están conectados de manera secuencial, implicando que el entrenamiento de uno de los clasificadores depende de los anteriores. Este tipo de técnica presenta la particularidad de que cada clasificador es entrenado teniendo en cuenta los errores cometidos por los clasificadores anteriores y dándoles más importancia.

En la figura 2.6 se puede ver una representación gráfica de este tipo de métodos, mostrando como se entrenan los clasificadores de forma secuencial, considerando las debilidades de los modelos anteriores. Notar que al igual que para el caso de métodos tipo bagging, los métodos tipo boosting se ven afectados por el problema de clases desbalanceadas, por lo que se suele combinar métodos de resampleo de datos entre cada iteración de entrenamiento para combatir el nivel de desbalance [25].

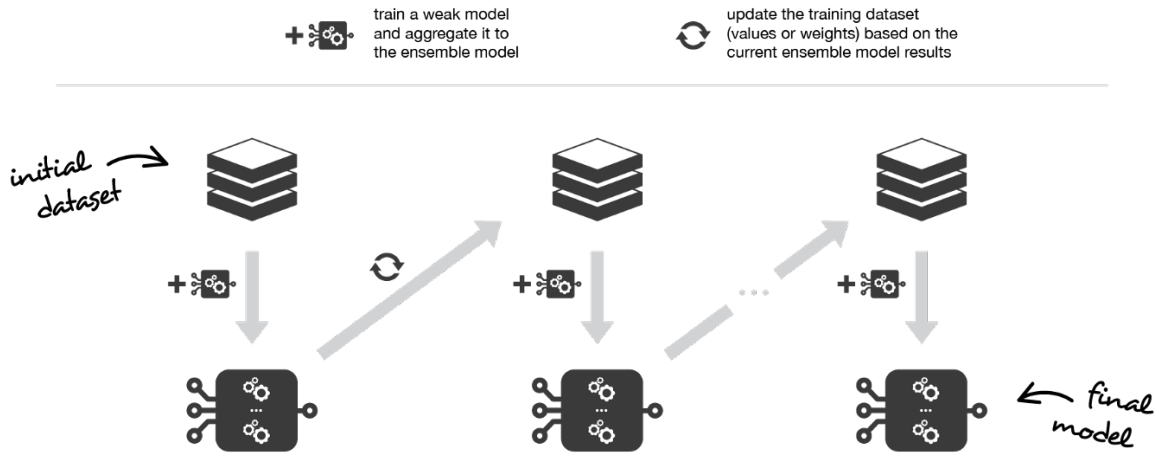

Figura 2.6: Representación gráfica algoritmo tipo boosting. [21]

A continuación se muestran algunos de los algoritmos de ensemble learning tipo boosting más importantes:

Easy Ensemble Classifier: Este clasificador es un ensamblaje tipo bagging, pero los clasificadores débiles consisten en clasificadores tipo boosting llamados AdaBoost [26], los cuales son entrenados con subconjuntos balanceados por el método de resampleo random under sampling. Por lo que esta técnica se beneficia de la combinación de los métodos tipo boosting, como de los métodos tipo bagging [27].

RUS Boost Classifier: Antes de llevar a cabo cada iteración del algoritmo tipo boosting, se aplica el método de resampleo random under sampling (RUS) al conjunto de entrenamiento, para

así balancear la cantidad de datos por clase. La ventaja de este método con respecto a algoritmos similares es que es más simple y el costo computacional asociado es menor, esto se debe a que el método de resampleo utilizado es bastante simple y al ser del tipo undersampling, la cantidad de datos va disminuyendo, lo que implica que el tiempo de entrenamiento sea más corto [25].

2.6. Redes neuronales

Las redes neuronales son un tipo de modelo de aprendizaje de máquinas inspirado en la estructura y el funcionamiento del cerebro biológico. Estas redes están compuestas por una serie de unidades llamadas nodos, que se conectan entre sí a través de enlaces. Estas herramientas son capaces de aprender a partir de datos de entrada, permitiendo que sean capaces de realizar tareas complejas, tal como la clasificación, predicción y regresión de datos.

Las redes neuronales se han utilizado con éxito en una amplia variedad de aplicaciones, incluyendo la visión computacional, el procesamiento del lenguaje natural, la detección de patrones y el diagnóstico de fallas en equipos industriales. Estas redes son particularmente útiles para abordar problemas que involucran grandes cantidades de datos y que son difíciles de modelar utilizando técnicas tradicionales.

Una red neuronal está compuesta por tres tipos de capas, la capa de entrada, las capas ocultas y la capa de salida. Con respecto a la primera, esta recibe directamente la información del problema, por lo que la cantidad de nodos presente corresponde a la cantidad de variables del problema. Por otro lado, las capas ocultas procesan y transmiten la información hasta entregarla a la capa de salida, la cual, dependiendo del tipo de problema, entrega la predicción de salida en diferentes formatos.

En la figura 2.7 se puede ver un ejemplo gráfico de una red neuronal, en donde se aprecia como se conectan cada una de las neuronas artificiales, los tres tipos de capas y en que dirección fluye la información.

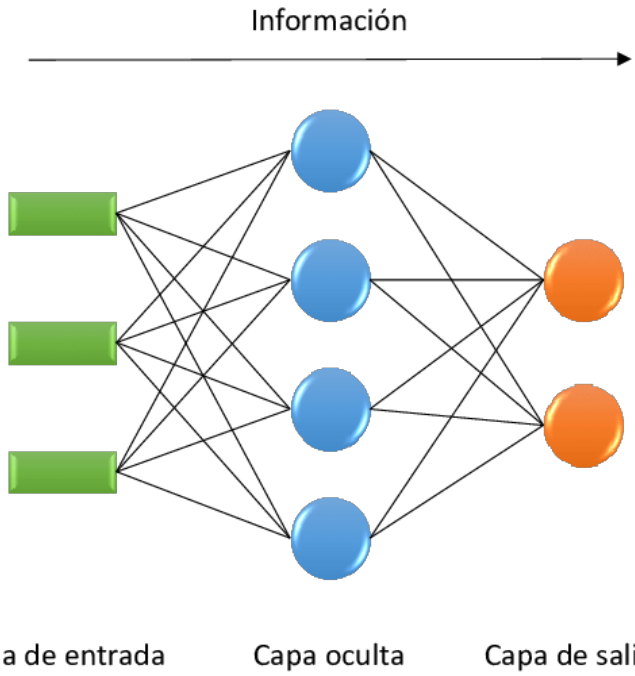

Figura 2.7: Representación gráfica red neuronal con una capa oculta. [28]

Además de los nodos y enlaces, las redes neuronales también están compuestas por parámetros entrenables, llamados pesos y sesgos, denotados w_i y b respectivamente, tal como se aprecia en la figura 2.8. Estos parámetros son utilizados por los nodos, los cuales reciben información de los nodos anteriores y emplea los parámetros para producir una salida que va hacia los siguientes nodos a los cuales está enlazado.

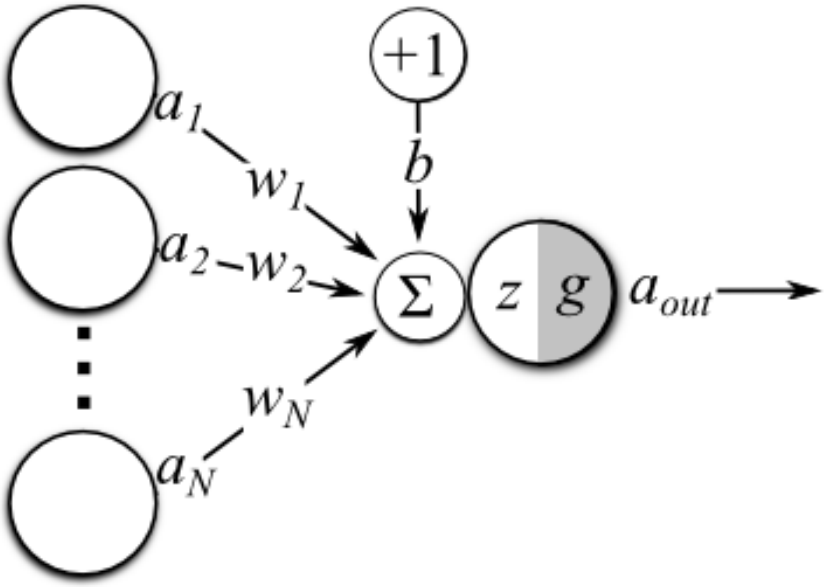

Figura 2.8: Representación pesos y sesgo en una red neuronal. [29]

Para calcular la salida de una neurona (a_{out}), primero se lleva a cabo una suma ponderada entre los datos de entrada (a_i) y los pesos (w_i), luego se suma el sesgo de la neurona (b). Por último, el

valor resultante es entregado a la función de activación ($g()$) de la neurona en cuestión, obteniendo así el valor de salida. Esto se puede representar matemáticamente mediante las siguientes fórmulas:

$$z = b + \sum_{i=1}^{N} a_i \cdot w_i \tag{2.2}$$

$$a_{out} = g(z) \tag{2.3}$$

Los pesos y sesgos de la red son ajustados durante el proceso de entrenamiento del modelo, esto con el fin de optimizar el funcionamiento de la red y así mejorar el rendimiento de esta para llevar a cabo la tarea encomendada. Para poder llevar a cabo el proceso de optimización de los parámetros entrenables se minimiza una función de costo que cuantifica la diferencia entre la predicción realizada y la predicción deseada.

Además de los pesos y los sesgos, están los hiperparámetros, los cuales son variables que no son modificados durante el proceso de entrenamiento, sino que son establecidas por el usuario y determinan las características generales del modelo. Algunos de los hiperparámetros más importantes son los siguientes:

- Optimizador, método mediante se ajustan los pesos y sesgos, algunos de los más utilizados son Adam, Sgd y Lbfgs.
- Número de capas ocultas.
- Fuerza del término de regularización L2 (α), el cual es un hiperparámetro de regularización que combate el sobreajuste al restringir el tamaño de los pesos.
- Función de activación ($g()$) para los nodos de las capas ocultas, algunas de las más utilizadas son Logistic, Tanh, Relu.
- Parada temprana, si es activada, tiene la función de terminar el entrenamiento cuando el desempeño no mejora en el conjunto de validación
- Número máximo de iteraciones, el cual aplica cuando el optimizador tiene problemas para converger, definiendo así un límite de iteraciones.

Realizar un ajuste de los hiperparámetros puede generar mejoras sustanciales, ya que al seleccionar la combinación óptima de hiperparámetos se puede maximizar la capacidad de predicción del modelo y ayudar a evitar el overfitting. Este proceso se puede llevar a cabo mediante el método de validación cruzada, el cual consiste en entrenar el modelo numerosas veces con diferentes combinaciones de hiperparámetros y evaluar su desempeño en el conjunto de validación. Luego de llevar a cabo el proceso anterior para diferentes conjuntos de entrenamiento y validación, se escoge la combinación de hiperparámetros que presenta un mejor desempeño promedio.

2.7. Métricas de evaluación

La correcta elección de las métricas de evaluación juega un rol crucial al momento de desarrollar un algoritmo. Tradicionalmente, el accuracy es la métrica más utilizada para estos fines, sin embargo, cuando se tiene el problema de clases desbalanceadas, esta deja de ser una métrica de evaluación adecuada, ya que la clase minoritaria tiene un impacto muy pequeño en comparación con la clase mayoritaria. Por ejemplo, en un problema donde una clase minoritaria está representada por solo el 1 % de los datos de entrenamiento, un clasificador simple podría catalogar a todos los datos como parte de la clase mayoritaria y aun así obtener un accuracy del 99 % [3]. Por lo que esta métrica no es útil en las aplicaciones con distribución desequilibrada en sus datos y que la clase minoritaria es la de interés.

Debido a lo anterior, se vuelve sumamente importante utilizar métricas de evaluación pertinentes a lo que se quiere, en este caso, aquellas que se enfoquen en la clase minoritaria. Una forma de visualizar el resultado de la clasificación del algoritmo de manera fácil es con matrices de confusión, tal como se ve en la tabla 2.1. Esta tabla representa el caso de clasificación binaria, es decir, cuando solo hay 2 clases, en donde la positiva corresponde a la clase de interés, es decir, las fallas del equipo.

Tabla 2.1: Matriz de confusión

		Predicción	
		Negativo	Positivo
Real	Negativo	Verdadero negativo (VN)	Falso positivo (FP)
	Positivo	Falso negativo (FN)	Verdadero positivo (VP)

A continuación, se muestran algunas de las métricas de evaluación que se pueden construir mediante una matriz de confusión:

- $Accuracy = \frac{VP+VN}{FN+VN+FP+VP}$
- $Sensitivity/Recall = \frac{VP}{FN+VP}$
- $Specificity = \frac{VN}{FP+VN}$
- $G-mean = \sqrt{Sensitivity \cdot Specificity}$
- $Precision = \frac{VP}{FP+VP}$
- $F1-score = \frac{2 \cdot Recall \cdot Precision}{Recall+Precision}$

En donde, cada una representa un comportamiento distinto del algoritmo evaluado, por lo que es importante saber qué información proveen, ya que si se utiliza la métrica incorrecta es muy difícil optimizar el algoritmo. Por ejemplo, el accuracy corresponde a la razón de elementos clasificados

correctamente, por lo que depende fuertemente del desbalance entre clases y no sirve para evaluar correctamente un algoritmo en este caso.

La métrica sensitivity es la razón entre los datos clasificados correctamente como positivos y el total de positivos reales, en donde para el caso de estudio, representa la cantidad de datos clasificados correctamente como fallas en comparación con la cantidad de datos que realmente son fallas. Teniendo en cuenta lo anterior, se puede decir que esta métrica también sirve para cuantificar la capacidad de evitar falsos negativos, es decir, datos clasificados erróneamente como fallas.

Mientras tanto, la métrica specificity es la razón entre los datos clasificados correctamente como negativos y el total de negativos reales, por lo que esta métrica sirve también para cuantificar la capacidad del clasificador para evitar falsos positivos.

Cabe mencionar que generalmente hay una compensación entre las métricas sensitivity y specificity, ya que si se aumenta mucho una, la otra podría verse afectada negativamente [4]. Es por eso que se desarrollan métricas como G-mean, la cual toma en cuenta ambas métricas al calcular la media geométrica entre ambas, esto con el objetivo de maximizar el reconocimiento de cada una de las clases minoritarias y mayoritarias, manteniendo estas métricas equilibradas. Presentando la ventaja que es independiente de la distribución de datos entre clases.

Otra métrica útil para el caso de clases desbalanceadas es f1-score, la cual también toma en cuenta la compensación entre dos métricas, recall y precision, en donde la métrica precisión consiste en cuantos de los datos clasificados positivos son realmente positivos. La métrica f1-score consiste en la media armónica entre las métricas recall y precision, por lo que permite optimizar el clasificador de manera equilibrada, en otras palabras, para mejorar el f1-score deben mejorar tanto la métrica recall como precision.

Capítulo 3

Metodología

3.1. Montaje experimental

Como se mencionó en la introducción, cada sistema motor-bomba está equipado por un total de 16 sensores, lo cuales están ubicados en diferentes partes. En la figura 3.1 se puede ver una representación gráfica del sistema, mostrando como interactúan el motor y la bomba. En esta imagen, el motor corresponde al equipo azul y este es acoplado mediante un eje a la bomba, el cual es el equipo gris.

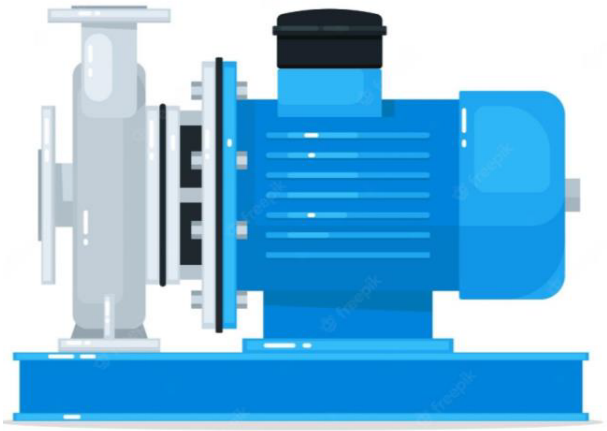

Figura 3.1: Ejemplo sistema motobomba.

A continuación, se detallan todas las variables físicas monitoreadas:

1. Fase 1 corriente motor.
2. Fase 2 corriente motor.
3. Fase 3 corriente motor.
4. Caudal bomba.
5. Temperatura descanso motor lado libre.

6. Temperatura descanso motor lado acople.
7. Temperatura descanso bomba lado libre.
8. Temperatura descanso bomba lado acople.
9. Proximidad descanso motor lado libre dirección x.
10. Proximidad descanso motor lado libre dirección y.
11. Proximidad descanso motor lado acople dirección x.
12. Proximidad descanso motor lado acople dirección y.
13. Proximidad descanso bomba lado libre dirección x.
14. Proximidad descanso bomba lado libre dirección y.
15. Proximidad descanso bomba lado acople dirección x.
16. Proximidad descanso bomba lado acople dirección y.

3.2. Caracterización de datos y falla de entrenamiento

El primer paso antes de abarcar este tipo de problemas es realizar una caracterización de los datos con que se está trabajando, es decir, ver detalladamente la naturaleza de los datos. Esto provee un mejor entendimiento de los datos y de la falla en sí, permitiendo tener una imagen más clara de como abordar el desafío.

El sistema de monitoreo de la bomba escogida recolectó datos desde el 13 de noviembre de 2018 hasta el 13 de marzo de 2019, lo que consiste en un total de 27.860 instancias. Cada instancia consiste en los valores medidos por los sensores instalados, los cuales miden 3 fases de corriente del motor, caudal entregado por la bomba, temperatura en 4 puntos del sistema y vibraciones en 8 puntos, lo cual suma un total de 16 variables que alimentan el modelo de decisión.

Para caracterizar los datos, se analizarán visualmente los historiales de monitoreo de cada una de las variables físicas, en donde se estudiará el comportamiento de cada una de estas a lo largo del tiempo. En específico, se analizan las características generales de cada variable, los posibles datos anómalos (detenciones máquina, mal funcionamiento de los sensores, entre otros) y el comportamiento de los datos cuando se presenta la falla en cuestión.

3.3. Preprocesamiento

Con el fin de facilitar la labor de los modelos a desarrollar, se vuelve imperante realizar un preprocesamiento de los datos. Para este caso de estudio, el preprocesamiento consta de 3 etapas:

eliminar datos correspondientes a las detenciones del equipo, aplicar método de la media móvil y normalizar los datos.

La presencia de datos correspondiente a detenciones podrían llevar al algoritmo a identificar tendencias inexistentes en la realidad, debido a esto, se manejan como si fueran datos anómalos y estos son eliminados. Esto se realiza bajo el criterio de que un dato corresponde a una detención cuando la corriente medida por los sensores es menor a cierto umbral. Además, con el fin de considerar solo datos representativos, se identifican los datos correspondientes al régimen transiente y luego se eliminan, es decir, cuando el equipo transiciona de un estado encendido a detenido o viceversa. En específico, se detecta una detención en los datos y se elimina una cierta cantidad de instancias antes y después de esta.

Cabe mencionar que las variables de corriente solo son utilizadas para determinar las detenciones, luego de esto, son descartadas junto con la variable del caudal, dejando solo como información de entrada para los clasificadores las variables de temperatura y vibración. Esto se debe a que las variables descartadas no aportan significativamente a caracterizar la falla escogida para entrenar los modelos. Sumado a lo anterior, esto ayuda a disminuir el tamaño de la base de datos, lo que implica una reducción en el costo computacional de entrenamiento de los modelos.

Luego, con el objetivo de pulir los datos recopilados por los próximetros, se aplica una resta de la media móvil para cada variable correspondiente a vibraciones. Cabe mencionar que la media móvil es recalculada después de cada detención, ya que en algunos casos, la media cambia abruptamente después de algunas detenciones.

Por último, se lleva a cabo una normalización de los datos, esto con el fin de asegurar un buen desempeño de los algoritmos de clasificación, ya que en algunos casos estos asumen que las diferentes variables están centrados en cero y tienen una varianza del mismo orden de magnitud. Por lo que si no se cumplen están condiciones, las variables con una varianza con un orden de magnitud más grande podrían dominar la función objetivo e impedir que el algoritmo aprenda de las otras variables de manera esperada.

En este caso, se aplica el método de normalización estándar de la librería de python scikit-learn para cada variable, la cual se calcula de la siguiente forma:

$$Z = \frac{x_i - \mu}{\sigma} \tag{3.1}$$

Donde Z es el valor del dato recalculado, μ es la media de los datos cada variable y σ es la desviación estándar.

3.4. Definición etiquetas y conjuntos

Hay diversas alternativas de enfoques al problema de diagnóstico de fallas, pero en este caso particular se decide el uso de algoritmos de clasificación, los cuales consistes en técnicas de aprendizaje supervisado. En este tipo de técnicas, se intenta deducir una función a partir de datos de entrenamiento, lo que para el caso de estudio consiste en entrenar un modelo con datos de entrenamiento, para que este sea capaz de clasificar si los datos corresponden a una falla o no.

Para lograr entrenar este tipo de algoritmos (clasificadores), se requiere de una base de datos etiquetada, lo que en este caso se traduce en que cada uno de los datos estén etiquetados si corresponden a datos saludables o a una falla del equipo. Este se debe a que durante el proceso de entrenamiento, el algoritmo debe ser capaz de corregir los errores cometidos, para así ir mejorando el límite de decisión y su capacidad de predicción.

Usualmente, en los casos de estudio de índole académica (experimentos controlados), las bases de datos están etiquetados, pero cuando se trabaja con datos reales de la industria, es común que estos no estén etiquetados, lo que dificulta el desarrollo de los algoritmos de aprendizaje de máquinas. Para el caso de estudio en cuestión, se tiene que los datos no están etiquetados, ya que solo se tiene información de cuando se detuvo la máquina.

El criterio para etiquetar los datos correspondientes a la falla consistió en tomar el instante en donde se detuvo la máquina y todos los datos aproximadamente 1 día antes fueron considerados como parte de la falla. Esto se realiza con el objetivo de asegurar que se abarque toda la falla y que el algoritmo de clasificación sea capaz de detectar patrones en los datos justo antes de que ocurra la falla, detectando síntomas previos.

Por otro lado, también es sumamente importante la definición de los conjuntos de entrenamiento y testeo para el desarrollo de los algoritmos de clasificación. Esto se debe a que estos deben ser escogidos de tal forma que la evaluación del algoritmo sea lo más similar a la realidad, es decir, en caso de que el modelo sea encargado de detectar fallas de una máquina en tiempo real. Para lograr esta similitud, se toma el conjunto de datos definido como fallas y se escoge una ventana de tiempo dentro de esta para sea parte del conjunto de testeo.

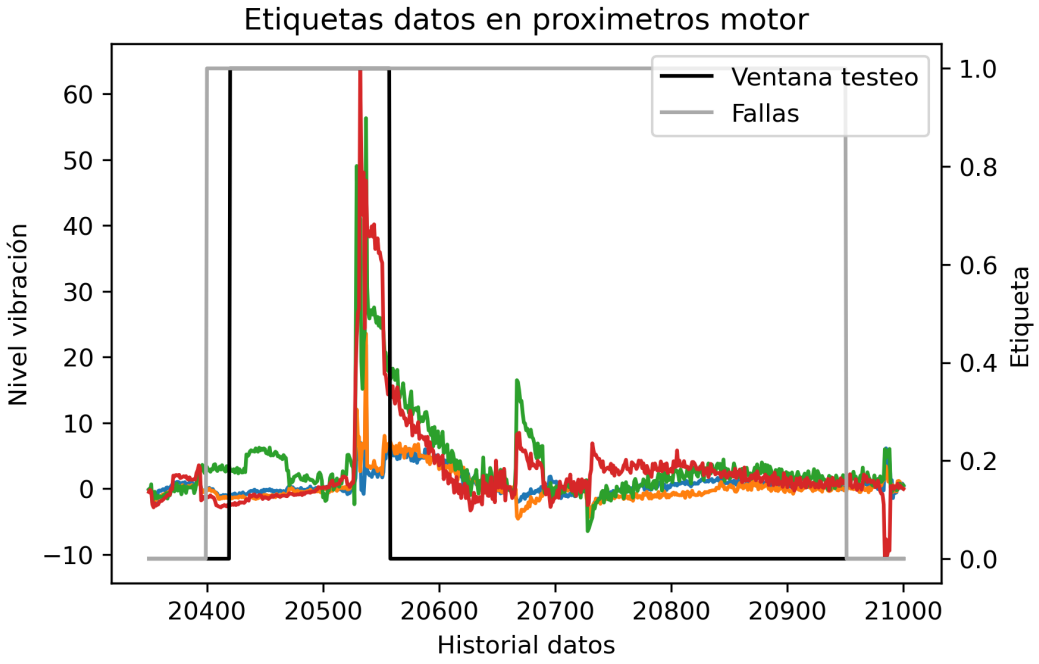

Figura 3.2: Visualización gráfica de las etiquetas en vibraciones del motor.

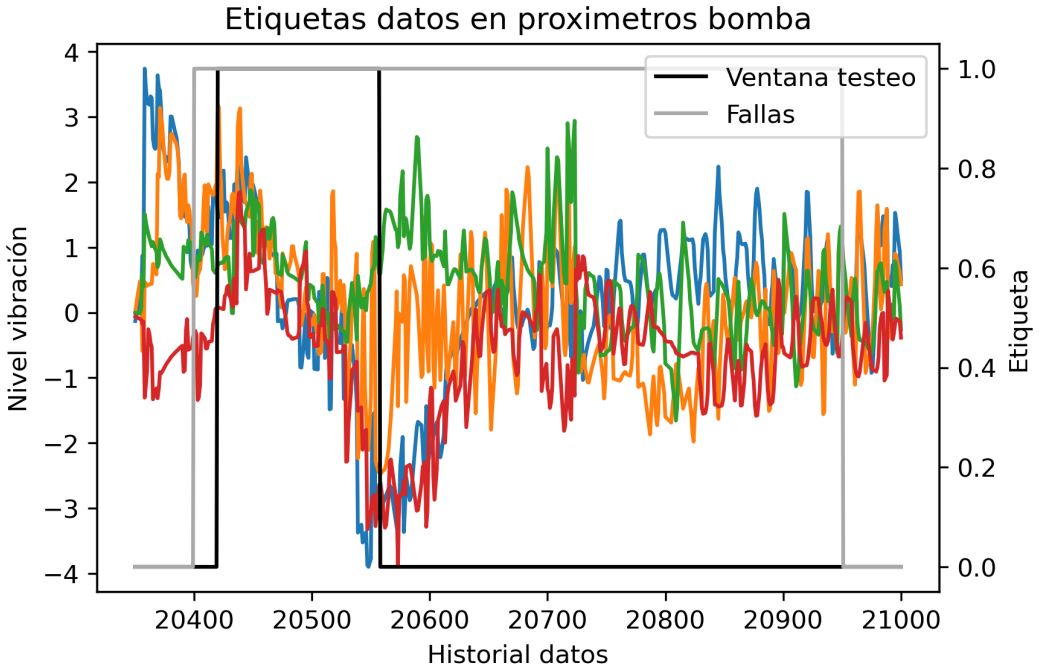

Figura 3.3: Visualización gráfica de las etiquetas en vibraciones del bomba.

En las figuras 3.2 y 3.3, se puede ver un acercamiento de la falla en el historial de datos, además de 2 escalones que representan los datos etiquetados como falla y aquellos que fueron asignados al conjunto de testeo. Cabe mencionar que se escogen esos datos para el conjunto de testeo, ya que tienen información crucial sobre la falla y porque están situados al inicio de esta, por lo que al evaluar el algoritmo, se verá que tan bien detecta este tipo de fallas antes de que sucedan o muestren

síntomas severos.

El ancho de esta venta de tiempo se define como el 25 % de los datos etiquetados como fallas, ya que se toman el 25 % de todos los datos para evaluar el algoritmo. En otras palabras, el 75 % de los datos son parte del conjunto de entrenamiento y el 25 % restante es parte del conjunto de testeo. A continuación se detallan los pasos generales para definir cada conjunto:

1. Determinar ventana de tiempo dentro de conjunto de datos con falla, tal que corresponda al 25 % de datos del conjunto. Esta ventana de tiempo es asignada al conjunto de testeo.
2. Asignar datos restantes del conjunto de fallas al conjunto de entrenamiento.
3. Se asignan aleatoriamente datos del conjunto saludable al conjunto de testeo, hasta que este corresponda al 25 % de los datos totales.
4. Asignar todos los datos restantes al conjunto de entrenamiento.

3.5. Selección y desarrollo de algoritmos

Una vez terminado el preprocesamiento de los datos y la definición de los conjuntos de entrenamiento y testeo, prosigue definir y entrenar el algoritmo de aprendizaje de máquinas para clasificar los datos. En este paso es donde se aplican los enfoques para atenuar el problema de clases desbalanceadas que se abordaron en el capítulo de antecedentes.

Cabe mencionar que para todos los métodos de resampleo de datos desarrollados se utiliza el mismo clasificador, el cual consiste en una red neuronal multicapa (RNM) de la librería sklearn [30]. Esto se debe a que se están comparando diferentes métodos de resampleo de datos, por lo que todos los otros pasos del desarrollo del algoritmo deben ser iguales. Además de esto, se escoge ese clasificador en específico porque suele presentar problemas cuando se tiene una distribución desequilibrada entre clases, por lo que permite ver si realmente mejora el rendimiento al combinarlo con métodos de resampleo.

Con el fin de evaluar si los algoritmos aplicados realmente ayudan a abordar el problema de clases desbalanceadas, se vuelve necesario evaluar también el caso en donde no se agrega ningún algoritmo. Este caso base consiste en tomar los datos, preprocesarlos y luego aplicar directamente el clasificador (RNM).

A continuación se enumeran los algoritmos desarrollados y evaluados según cada tipo de enfoque para solucionar el problema de clases desbalanceadas:

1. RNM (caso base).

Con respecto a los métodos de resampleo de datos tipo oversampling, se tienen los siguientes:

1. Random Over Sampler + RNM.
2. SMOTE + RNM.
3. ADASYN + RNM.
4. Borderline SMOTE + RNM.
5. SVMSMOTE + RNM.

Por otro lado, se desarrollan los siguientes métodos de resampleo de datos tipo undersampling

1. Cluster Centroids + RNM.
2. Condensed Nearest Neighbour + RNM.
3. Edited Nearest Neighbours (ENN) + RNM.
4. Repeated Edited Nearest Neighbours + RNM.
5. All-KNN + RNM.
6. Near Miss + RNM.
7. Neighbourhood CleaningRule + RNM.
8. One Sided Selection + RNM.
9. Random Under Sampler + RNM.
10. Tomek Links + RNM.

Luego, para aprovechar las ventajas de los métodos de oversampling y undersampling, estos se combinan en los siguientes métodos:

1. SMOTE + ENN + RNM.
2. SMOTE + Tomek Links + RNM.

Con respecto al enfoque de ensemble learning, se desarrollan dos métodos del tipo bagging y dos del tipo boosting, en donde los dos primeros corresponden al tipo bagging. A continuación se listan los métodos en cuestión:

1. Balanced Bagging Classifier.
2. Balanced Random Forest Classifier.
3. RUS Boost Classifier.
4. Easy Ensemble Classifier.

Cabe mencionar que todos los modelos están implementados en la librería de python imbalanced-learn [31].

3.6. Visualización gráfica de métodos de resampleo

Uno de los obstáculos más grandes a la hora de clasificar datos es la superposición de las clases, lo que para este caso se puede visualizar claramente en un gráfico de dispersión. En donde ambas clases se pueden representar como puntos de diferentes colores, permitiendo apreciar los efectos de la aplicación de los métodos de resampleo en cuestión.

Este gráfico es obtenido al aplicar el método de análisis de componentes principales (PCA), el cual permite tomar el espacio de doce dimensiones, una por cada variable física utilizada, y llevarlo solo a dos dimensiones. Esta transformación permite visualizar los datos de manera fácil, en donde los ejes consisten en las dos componentes principales obtenidas.

Cabe mencionar que primero se aplica el método de resampleo en el espacio de doce dimensiones y luego se aplica PCA para llevarlo a dos dimensiones y poder visualizar los resultados, por lo que las componentes principales obtenidas pueden variar entre métodos de resampleo. Debido a esto, algunos gráficos de dispersión pueden ser ligeramente distintos.

3.7. Selección de hiperparámetros

La optimización de los hiperparámetros es sumamente importante al momento de entrenar un modelo de aprendizaje de máquinas, ya que estos juegan un papel significativo en la eficiencia del modelo, en específico, en su capacidad de generalización. Los hiperparámetros son variables que no se ajustan durante el proceso de entrenamiento, sino que son establecidos por el usuario y determinan las características generales del modelo, tales como su estructura.

En el caso de las redes neuronales, existen diversos hiperparámetros que se pueden ajustar, incluyendo el número de capas ocultas, el número de neuronas en cada capa, el tipo de función de activación empleada, la fuerza del término de regularización L2, entre otros.

Esta optimización consiste en encontrar la combinación de hiperparámetros tal que el modelo presente un mejor rendimiento en la tarea encomendada. Este proceso es llamado ajuste de hiperparámetros y consiste en escoger sistemáticamente diferentes combinaciones de hiperparámetros y evaluar el rendimiento del modelo en un conjunto de datos de validación.

Para llevar a cabo esto, hay diversas herramientas ya implementadas por librerías de Python, tal como RandomizedSearchCV de sklearn [32], la cual utiliza el método de validación cruzada para evaluar cuál es la combinación de hiperparámetros óptima. Cabe mencionar que esta herramienta no realiza una evaluación exhaustiva de todas las combinaciones, sino que realiza una evaluación de combinaciones escogidas aleatoriamente dentro de las opciones dadas por el usuario, lo que disminuye considerablemente el costo computacional.

A continuación, se muestran los hiperparámetros a ajustar y los posibles valores que estos pueden tomar:

- Número de capas ocultas: 10, 15, 20, 30, 35 o 40.
- Fuerza del término de regularización L2 (Alpha): 0.0001, 0.001, 0.01 o 0.1.
- Función de activación: logistic, tanh o relu.
- Parada temprana o early stopping: Verdadero o Falso.
- Número máximo de iteraciones: 200, 500 o 1000.

Cabe mencionar que el número de capas ocultas predefinido es de 100, pero con el fin de simplificar el modelo para evitar un sobreajuste, se limita a un número más bajo.

3.8. Evaluación

La evaluación de los algoritmos desarrollados consta de dos etapas, la evaluación al separar la base de datos original en subconjuntos de datos de entrenamiento y testeo, y la evaluación de la capacidad de generalización del mejor modelo entrenado en la primera etapa al entregarle una base de datos completamente nueva.

Debido a que el proceso de generar los conjuntos de entrenamiento y testeo en la primera etapa de evaluación tiene una componente aleatoria, cada combinación es distinta. Para atenuar el efecto de esta componente aleatoria en los resultados, se vuelve necesario realizar el proceso de evaluación numerosas veces y luego calcular la media y desviación estándar de los resultados.

Además de lo anterior, también es importante tener en consideración esta componente aleatoria si es que se quiere que todos los modelos sean evaluados bajo las mismas condiciones. Es por esto que para cada combinación generada de conjuntos de entrenamiento y testeo, todos los algoritmos son evaluados. Esto permite que los resultados obtenidos no se vean afectados por variables externas y que el rendimiento mostrado por los algoritmos sea comparable.

Por otro lado, para la segunda etapa de evaluación se toma una base de datos que consiste en el historial de monitoreo de otro equipo igual al que se entrenaron los modelos anteriores y que contiene una falla. Luego, se toma el modelo con mejor desempeño en la primera etapa de evaluación y se evalúa respecto a la falla nueva, cuantificando así la eficiencia de este para diagnosticar fallas nuevas, en otras palabras, su capacidad de generalización.

Capítulo 4

Resultados y discusión

4.1. Caracterización de datos y falla de entrenamiento

A continuación, se muestra el historial de monitoreo de cada una de las variables que alimentan el modelo desarrollado, analizando su comportamiento a lo largo del tiempo y como se ven afectados los datos al momento de presentarse la falla.

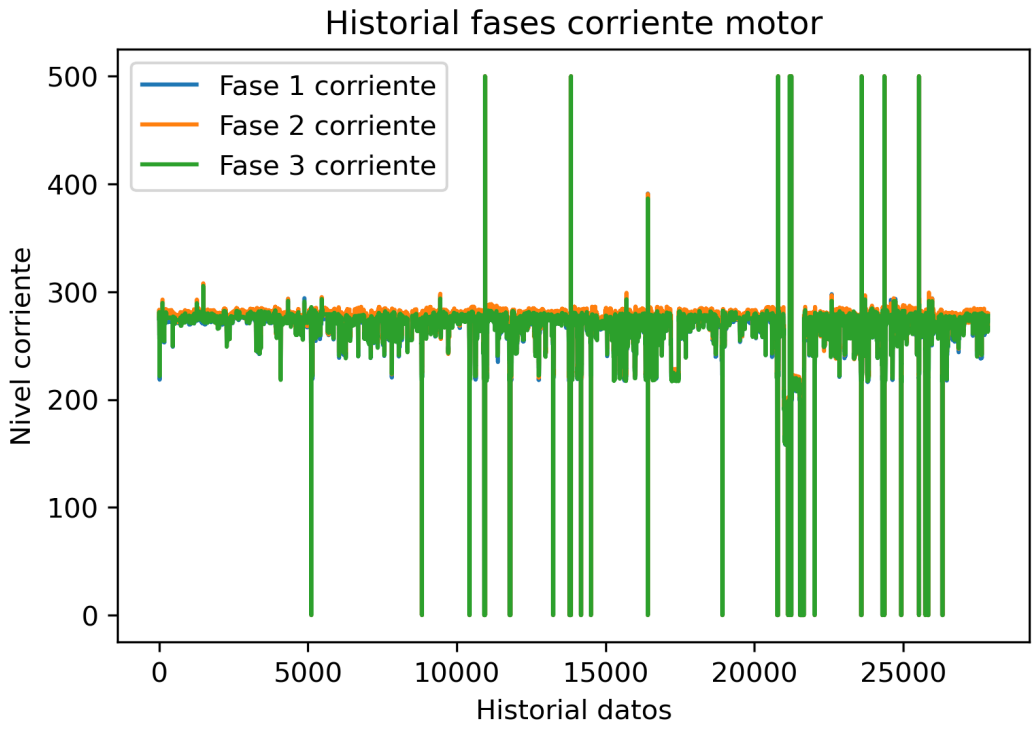

Figura 4.1: Historial de monitores de las 3 fases de corriente del motor.

En la figura 4.1 se puede observar la corriente a lo largo del tiempo, destacando que las 3 fases están altamente correlacionadas, por lo que las curvas se superponen. En general, los valores de corriente no cambian mucho a lo largo del tiempo, pero se pueden ver numerosos altos y bajos bruscos en las curvas, las cuales representan cuando la máquina está detenida, esto puede ser una

buena herramienta para determinar cuando hubo una detención.

Además de lo anterior, se puede observar que alrededor del dato 21.000 del historial de monitoreo (eje de las abscisas), hay una disminución brusca en el nivel de corriente que se mantiene por un tiempo más prolongado. Esto se debe a que hubo una detención de la máquina después de la falla estudiada, a pesar de esto, no se ve un gran cambio en los datos debido a la falla en sí.

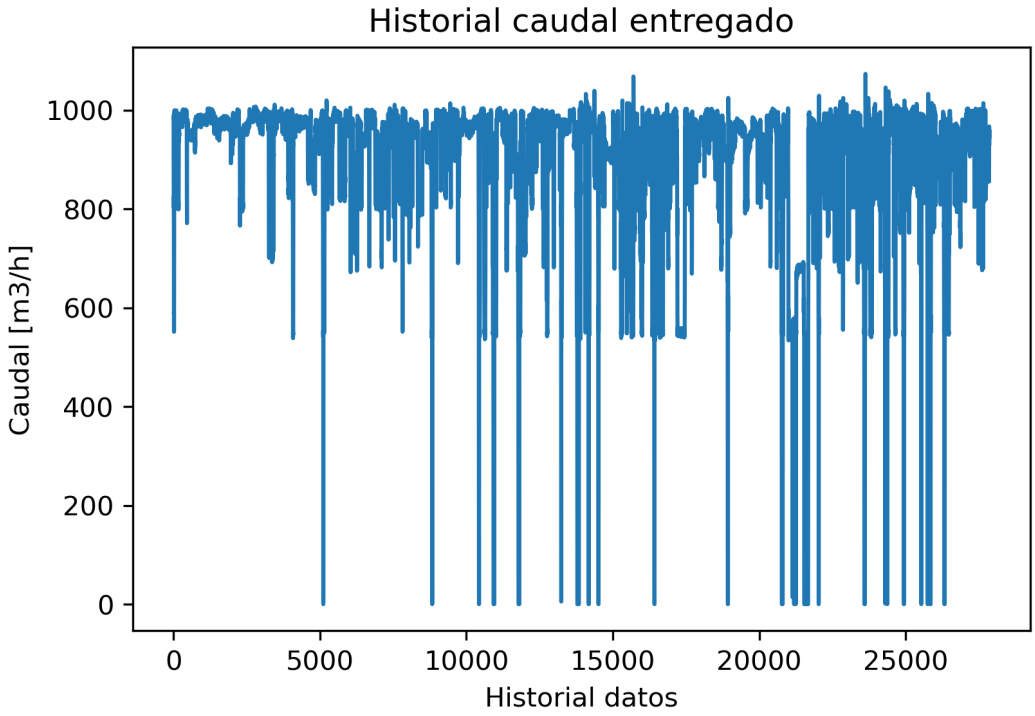

Figura 4.2: Historial de monitoreo del caudal entregado por el sistema.

Por otro lado, la figura 4.2 muestra el historial de monitoreo del caudal entregado del sistema, en donde también se puede ver claramente los momentos cuando la máquina está detenida. Cabe mencionar también que el régimen mínimo de impulsión del sistema es de 680 $[m^3/h]$, lo cual se puede ver en que la mayoría de los datos que no corresponden a detenciones son mayor a ese umbral. En los casos que el sistema no puede llegar a ese umbral, la máquina debe ser detenida e inspeccionada.

Al igual que en el caso de la corriente, se puede observar que hay una gran detención, pero visualmente no se ven cambios que muestren la falla, por lo que la variable de caudal no se le entrega al modelo de decisión.

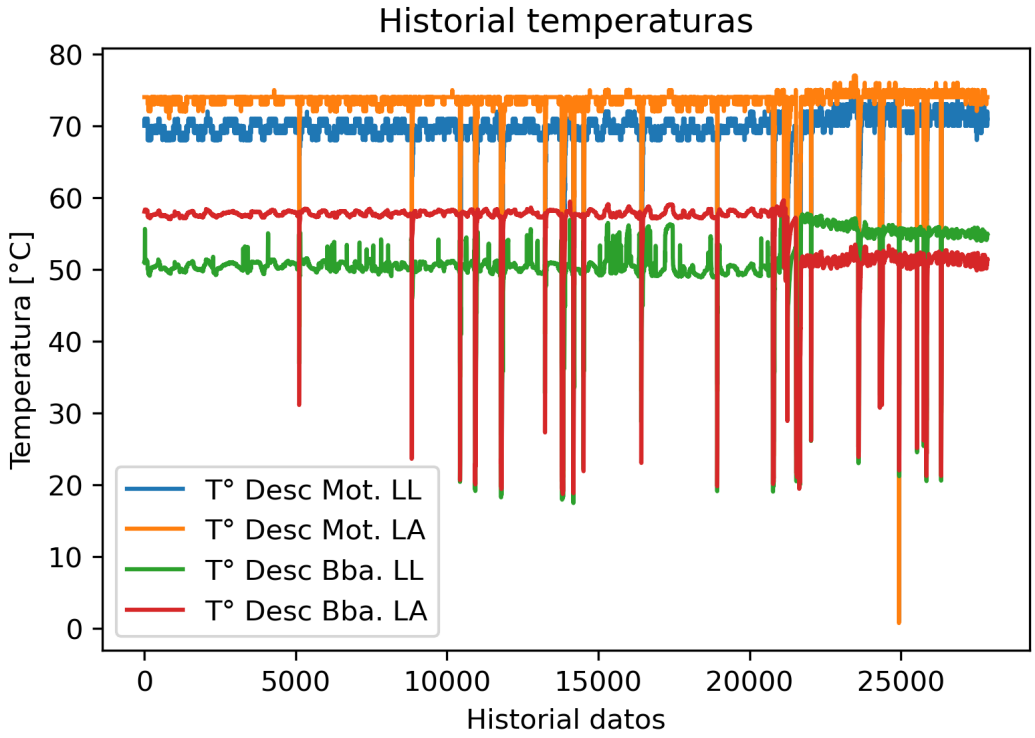

Figura 4.3: Historial de monitoreo de la temperatura en 4 puntos.

Con respecto a las temperaturas, en general estas tienen poca variación a lo largo del tiempo, sin considerar las detenciones. Cada lugar medido del sistema muestra un rango de temperaturas distinto, lo que se puede deber al contexto, ya que ambos registros de temperatura del motor son superiores a los de la bomba. Posiblemente, esto es debido a que el agua que fluye a través de la bomba actúa como refrigerante y disminuye la temperatura de los elementos mecánicos cercanos.

Además de lo anterior, se puede ver que alrededor del registro 21.000 del historial de monitoreo hubo un cambio drástico en las temperaturas de la bomba, lo que muestra los primeros indicios de una falla en el equipo. En específico, se tiene que la temperatura medida en el lado libre del descanso de la bomba aumenta 10 [^oC] aproximadamente, mientras que la temperatura en el lado del acople del descanso de la bomba disminuye 10 [^oC] aproximadamente.

Por otro lado, las dos temperaturas medidas en el motor no presentan grandes cambios que indiquen las presencia de una falla. A pesar de esto, ambas variables se utilizan para alimentar el modelo de decisión.

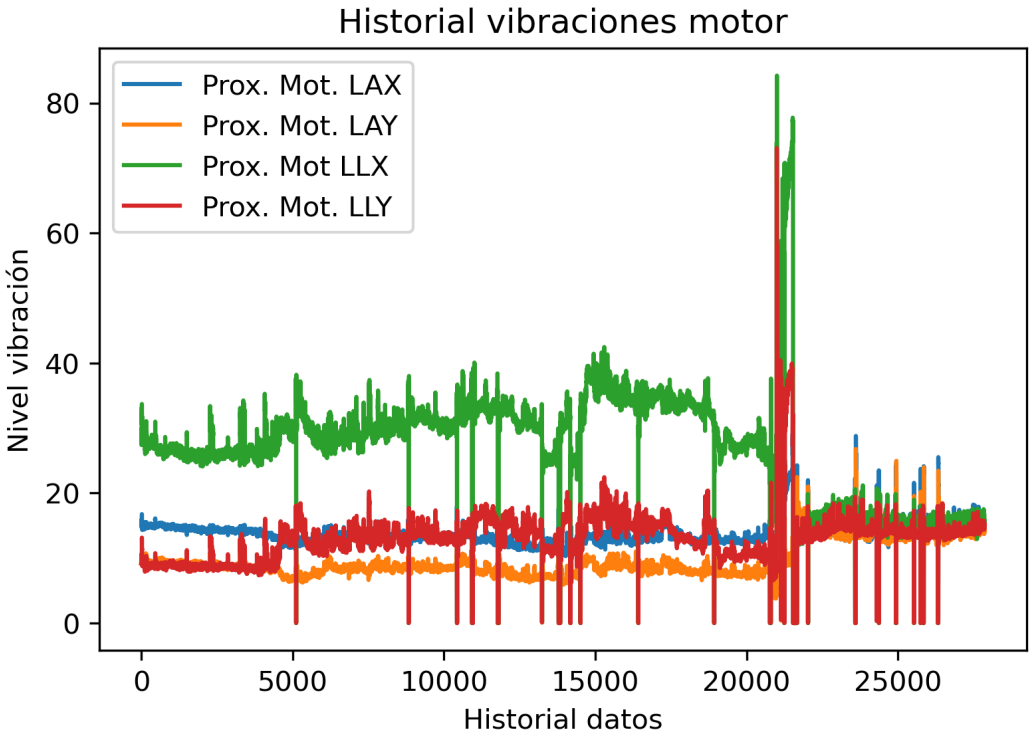

Figura 4.4: Historial de monitoreo del nivel de vibraciones del motor.

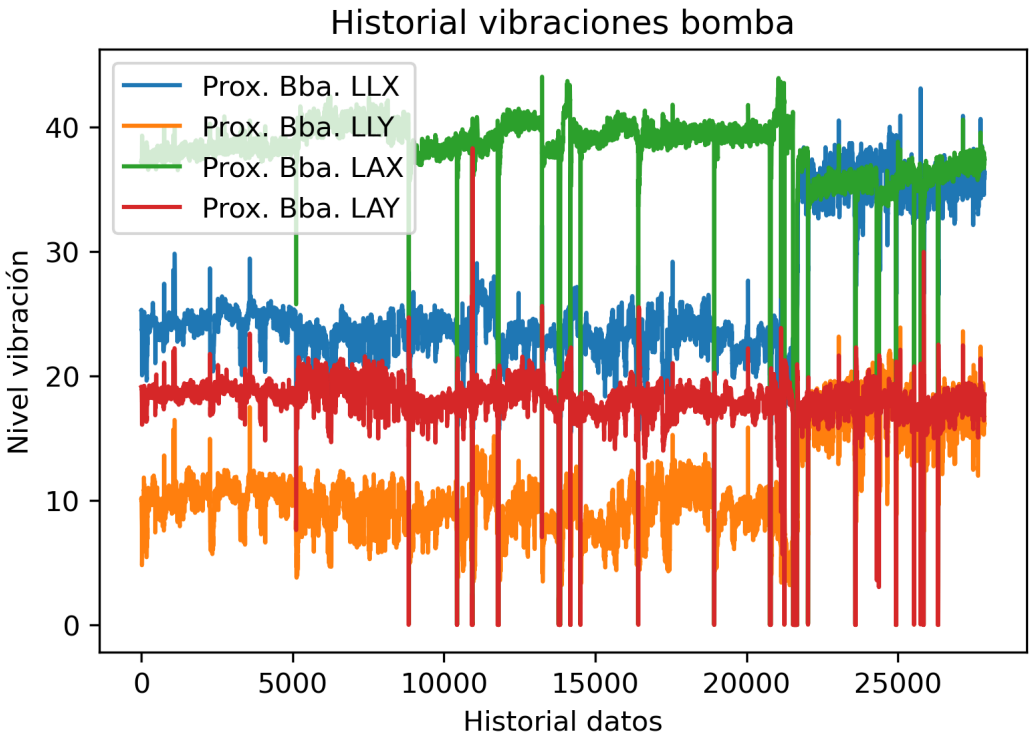

Figura 4.5: Historial de monitoreo del nivel de vibraciones de la bomba.

En las figuras 4.4 y 4.5 se puede observar los datos recolectados a lo largo del tiempo por los

sensores de proximidad instalados tanto en el motor, como en la bomba. Con respecto al historial del motor, al inicio no hay una gran variación a lo largo del tiempo, solo se presentan fluctuaciones a corto plazo, las cuales son de esperar en este tipo de variables físicas, ya que cualquier detalle o desajuste en el sistema mecánico puede generar cambios en el nivel de vibraciones.

A pesar de eso, en el mismo sector comentado en las variables anteriores se presenta un aumento drástico en los niveles de vibraciones del motor, lo que consiste en un claro síntoma de una falla en el equipo. Luego de la detección de este aumento en las vibraciones del motor, el equipo a cargo de la mantención decidió detener la máquina y revisar su estado de salud. Después de este punto, se puede ver como vuelve a la normalidad cada una de las variables, con algunas diferencias leves.

Por otro lado, con respecto a las vibraciones de la bomba se tiene que al inicio todas las variables se mantienen en su nivel normal, el cual es diferente para cada caso. Pero al igual que en los otros casos, los datos muestran un cambio brusco que representa la falla, detención y nueva puesta en marcha del equipo.

Vale la pena destacar qué gran parte de las variables sufrieron un cambio en su comportamiento luego de que el sistema mecánico fuera intervenido, en especial en el nivel promedio medido por los sensores. Esto se puede deber a que cualquier mínimo cambio en el modo en que es instalado el equipo puede afectar el comportamiento de las variables físicas mediadas, lo que podría dificultar el entrenamiento de modelos de diagnóstico de fallas, ya que el estado saludable de cada máquina puede variar según sus condiciones específicas.

4.2. Preprocesamiento

En las figuras 4.6, 4.7 y 4.8 se puede ver claramente el efecto generado por el preprocesamiento, en donde se muestra el resultado del preprocesamiento en las variables de temperatura, vibraciones de la bomba y vibraciones del motor.

En estos gráficos se puede apreciar como se eliminan exitosamente los datos de las detenciones y como se normalizan los datos. Por otra parte, se tiene que en el historial de temperaturas hay mediciones notoriamente bajas, las que pueden representar datos anómalos que no lograron ser eliminados por el preprocesamiento al cual fueron sometidos.

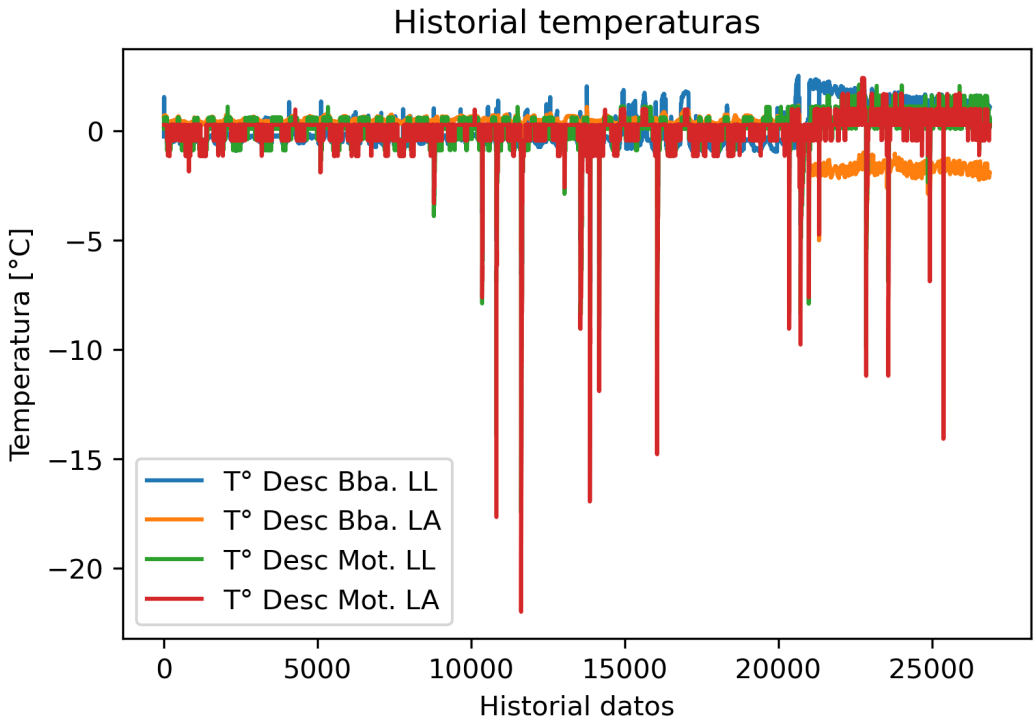

Figura 4.6: Temperaturas después del preprocesamiento.

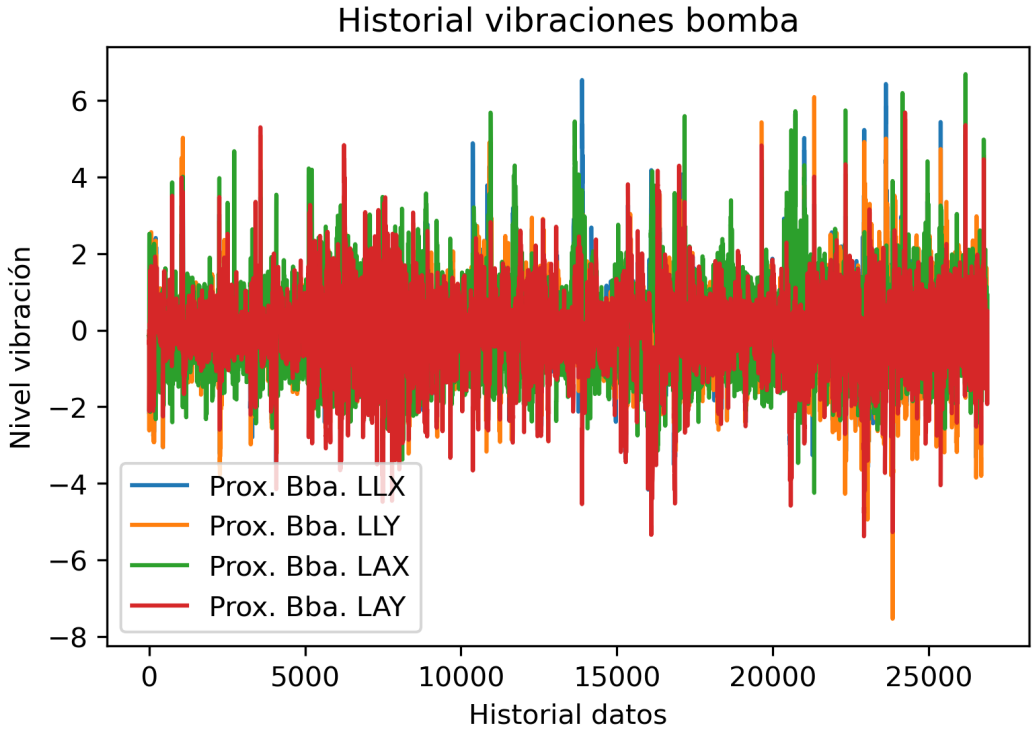

Figura 4.7: Nivel vibraciones bomba después del preprocesamiento.

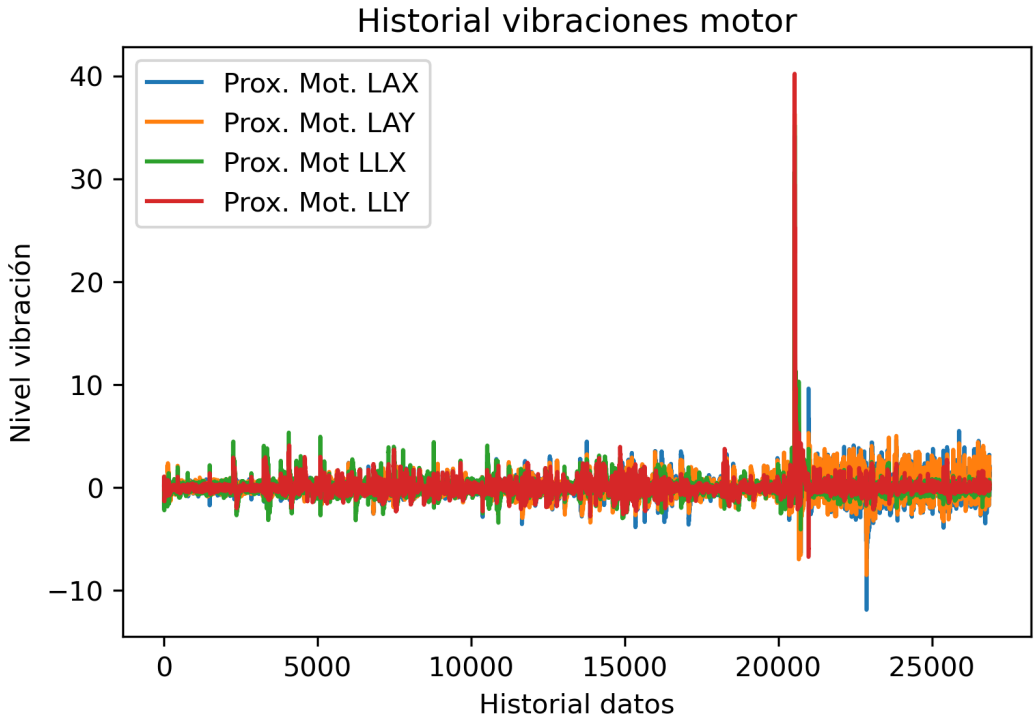

Figura 4.8: Nivel vibraciones motor después del preprocesamiento.

En adición a lo anterior, se tiene que la falla es fácil de identificar en el historial de vibraciones del motor, ya que presenta mediciones significativamente mayores en el momento de la falla. Por otro lado, en el historial de vibraciones de la bomba no hay indicios claros de la falla.

Considerando todo lo anterior, se puede afirmar que la falla utilizada para entrenar los modelos de decisión está fuertemente caracterizada por las vibraciones del motor, lo que puede dificultar al modelo la detección de una falla que esté caracterizada por otra variable física.

Cabe mencionar también que el tamaño de la base de datos disminuye levemente, pasando de 27.860 instancias a 26.881, lo que permite que el entrenamiento de los algoritmos de clasificación tenga un costo computacional ligeramente menor.

4.3. Visualización gráfica de clases en el caso base

En la figura 4.9 se tiene el gráfico de dispersión de la base de datos después del preprocesamiento y sin la modificación de ningún método de resampleo, en donde los puntos rojos corresponden a datos que representan fallas y los verdes representan cuando la máquina está saludable.

En general, el nivel de superposición presente es normal para este tipo de estudios, pero, en parte, se debe a que la definición de los datos con fallas fue de manera manual, por lo que es difícil discernir de manera clara cuando un dato corresponde a una clase u otra, esto resulta en que exista

una superposición entre estas.

Es importante considerar la naturaleza del método para visualizar los datos, ya que el algoritmo de aprendizaje de máquinas es entrenado en un espacio de doce variables, las que son transformadas mediante PCA para poder visualizar la información en dos dimensiones. Esto implica que el nivel de superposición entre clases no es igual en el espacio de doce dimensiones que en el de dos. Sin embargo, el gráfico de dispersión si sirve para tener una referencia de la superposición entre las clases estudiadas.

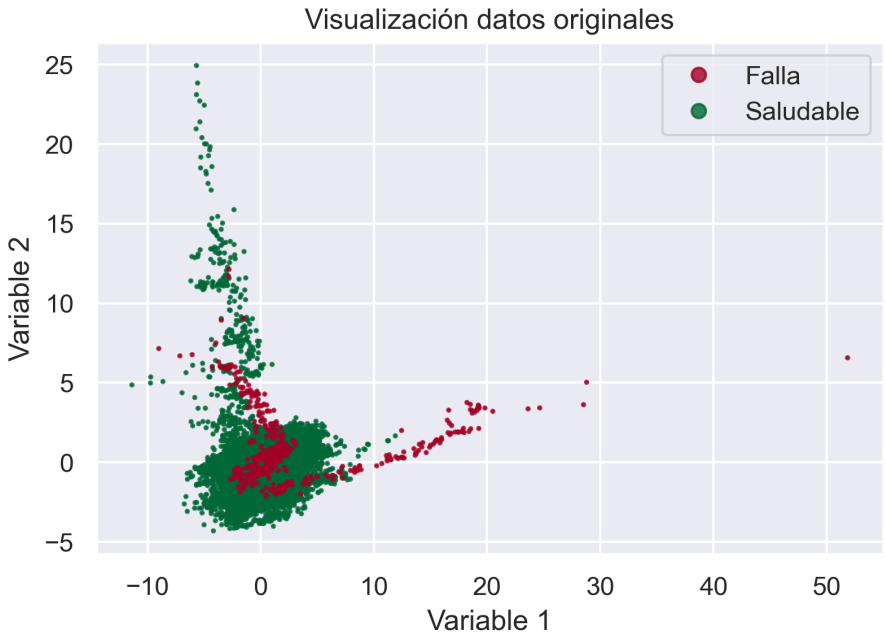

Figura 4.9: Visualización datos originales en dos dimensiones.

4.4. Evaluación del desempeño de los modelos

4.4.1. Resumen de los resultados de evaluación

En las figuras 4.10 y 4.11 se presenta un resumen del desempeño de todos los modelos evaluados, los cuales se obtienen al evaluar los modelos para una decena de combinaciones distintas de conjuntos de entrenamiento y evaluación. En donde posteriormente se calcula la media y desviación estándar de los valores de f1-score de testeo arrojados.

En específico, en la figura 4.10 se aprecia que para este caso particular, no todos los modelos ayudan a mitigar el efecto del problema de clases desbalanceadas, ya que muchos de los modelos implementados presentan un peor desempeño que el caso base, lo que los hace inútiles para esta aplicación.

Con respecto a los métodos de oversampling implementados, resulta que aquellos con criterios más simples para generar instancias nuevas de la clase mayoritaria, fueron los que mejor desempeño presentaron. Estos consisten en los métodos Random Over Sampler y SMOTE, en donde ambos generaron una mejora en el desempeño con respecto al caso base. Además de los modelos mencionados, el método SVM-SMOTE también presenta una mejora en su desempeño en comparación al caso original, mientras que los métodos de oversampling restantes presentan una empeora.

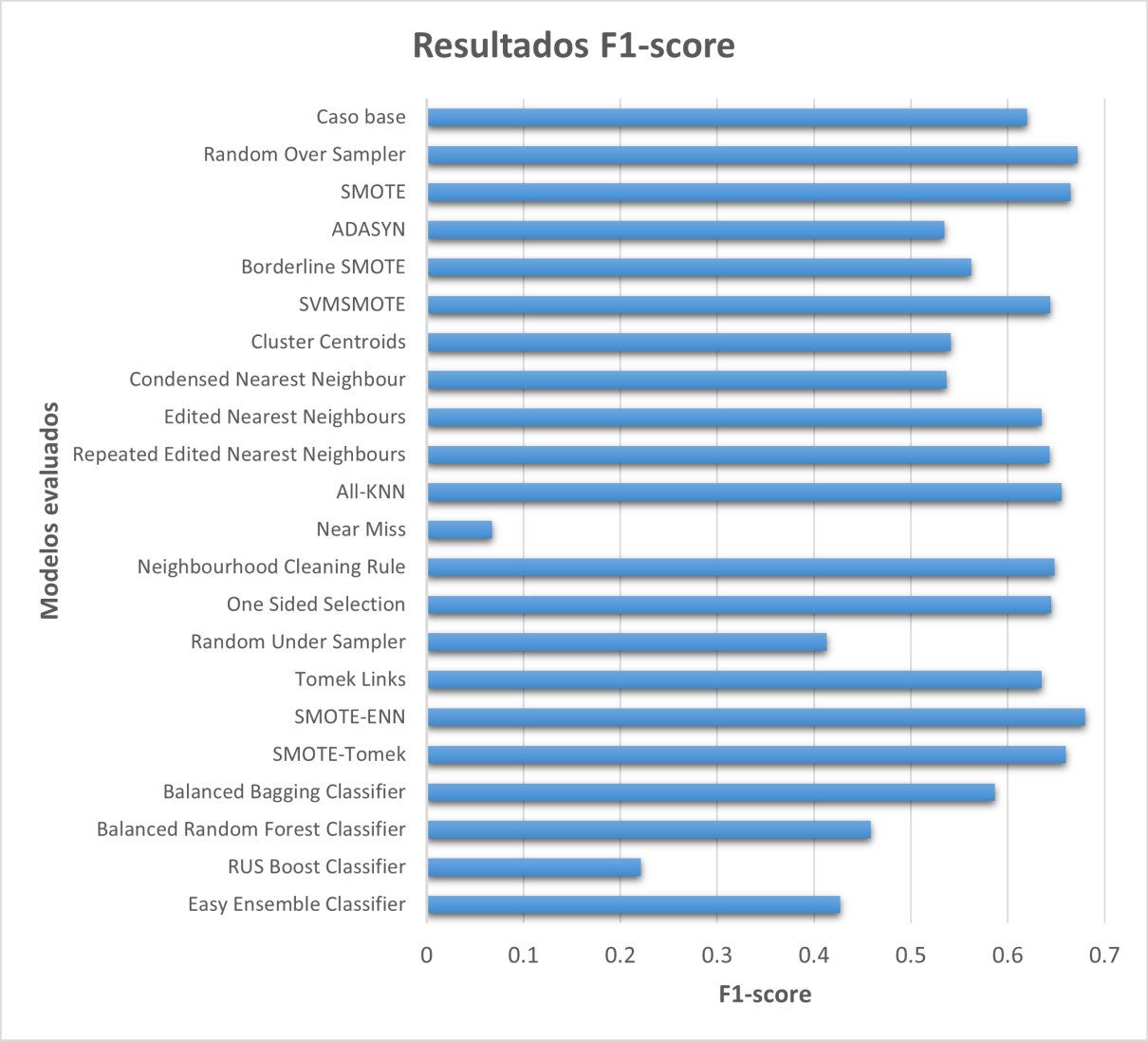

Figura 4.10: Resumen f1-score testeo modelos.

Por otra parte, referente a los métodos de undersampling, hay seis métodos que mejoran ligeramente su rendimiento en comparación al caso base, destacando All-KNN y Neighbourhood Cleaning Rule, los cuales tienen las mejores métricas de la categoría. Ambos métodos se enfocan en eliminar instancias cercanas al límite de decisión, lo que no siempre balancea las clases para que queden con la misma cantidad de datos. Sin embargo, permiten mantener la naturaleza intrínseca de los datos al mismo tiempo que se desecha información ambigua que confunde los clasificadores.

Con respecto a los métodos de undersampling que presentan un déficit en su rendimiento, todos tienen en común que eliminan una gran cantidad de datos, lo que se puede ver claramente en sus gráficos de dispersión, correspondientes a las figuras A.4, A.5, A.8 y A.10. Esto insinúa que al eliminar un gran volumen de datos, también se desecha de información crucial para representar el problema, lo que dificulta el posterior entrenamiento de algoritmos de aprendizaje de máquinas.

Acerca de los métodos que combinan técnicas de oversampling y undersampling, ambos presentan una mejora en su rendimiento, especialmente SMOTE-ENN, el cual obtiene los mejores resultados en comparación a todos los modelos evaluados. Lo que muestra que la combinación adecuada de diferentes técnicas de resampleo puede aprovechar las cualidades de cada una, ayudando a abordar el problema de clases desbalanceadas.

Por otro lado, los métodos de ensamblaje tipo bagging y tipo boosting presentan un desempeño inferior al caso base. El mejor de ellos es el modelo Balanced Bagging Classifier, de tipo bagging, el cual utiliza el método de undersampling Tomek links para combatir el problema de clases desbalanceadas.

A pesar de ser el método de ensamblaje con mejor desempeño, este sigue siendo deficiente, lo que se puede deber a una posible baja anormal en alguna de las métricas que componen la métrica f1-score, es decir, recall o precision. Por ejemplo, si dentro de las fallas reales se clasifican como falla muchos datos que realmente son saludables, esto genera una métrica precision baja, implicado un f1-score bajo, a pesar de que la métrica recall no necesariamente lo sea.

Por otra parte, los métodos de ensamblaje restantes utilizan el método de undersampling Random Under Sampling, lo que explica el déficit en su desempeño, ya que al evaluar el modelo Random Under Sampling + RNM, este presenta resultados considerablemente peores que sus pares. Esto indica que el déficit en el desempeño se explica en gran medida al efecto del método de resampleo

Cabe mencionar que debido a la compleja naturaleza de los métodos de ensamblaje, es necesario combinarlos con métodos de resampleo con un costo computacional bajo. Es debido esto que a pesar de los malos resultados del método Random Under Sampling, este es combinado con los métodos de ensamblaje.

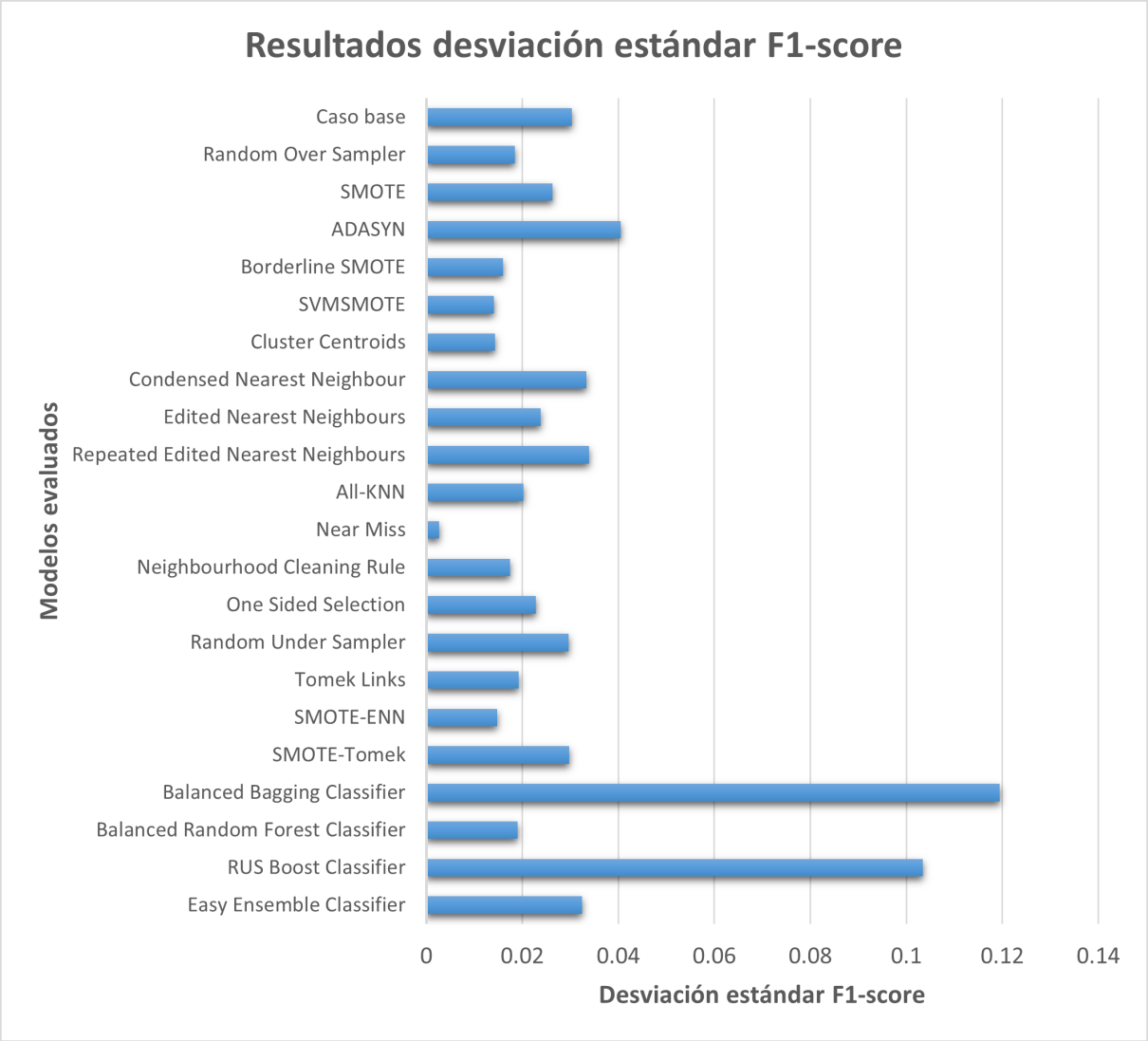

Figura 4.11: Resumen desviación estándar f1-score testeo modelos.

En cuanto a las desviaciones estándar de los modelos evaluados, expuestas en la figura 4.11, estas representan la estabilidad del modelo al cambiar la combinación de conjuntos de entrenamiento y evaluación, en donde mientras más baja la desviación estándar, más estable es el modelo, lo que es bueno si es que este presenta un nivel de f1-scores satisfactorios.

Vale la pena destacar la desviación estándar de los modelos de ensamblaje Balanced Bagging Classifier y RUS Boost Classifier, las cuales son notoriamente más altas que las de otros modelos. Para el caso del modelo Balanced Bagging Classifier, el cual presenta unos f1-score de testeo más cercanos al caso base, aunque se escoja la mejor iteración de los modelos evaluados para diferentes combinaciones de conjuntos, presentar poca estabilidad durante el entrenamiento no es una buena señal si es que se quiere utilizar el modelo para monitorear los equipos. Esto debido a que no se tiene certeza si realmente es un buen modelo o solo se obtuvieron buenos resultados en esa combinación porque cayó en un caso de overfitting.

En el caso contrario, modelos como Near Miss tienen una desviación estándar notoriamente baja, pero simultáneamente presentan un f1-score de entrenamiento muy bajo, lo que indica a que sufre de underfitting. El caso óptimo es un modelo el cual presenta un alto f1-score de testeo y una desviación estándar baja, lo que se cumple en el caso del modelo SMOTE-ENN.

Cabe mencionar que a pesar de que algunos modelos mejoren su rendimiento respecto al caso base, estos aún dejan mucho que desear, ya que presentar un f1-score de testeo del orden de magnitud de 0.68 no es lo óptimo y no asegura que sean eficientes a la hora de detectar fallas nuevas. Para determinar la utilidad de estos modelos se vuelve necesario evaluarlos con una falla nueva y ver si es que es detectada.

Debido a la gran cantidad de modelos evaluados, solo se analizarán detalladamente los mejores de cada categoría, es decir, el caso base, los dos mejores métodos de oversampling, los dos mejores métodos de undersampling, el mejor método que combina oversampling con undersampling, el mejor método de ensamblaje tipo bagging y el mejor de ensamblaje tipo boosting. Estos se listan de manera ordenada a continuación:

- Modelo caso base.
- Modelo Random Over Sampler + RNM.
- Modelo SMOTE + RNM.
- Modelo All-KNN + RNM.
- Modelo Neighbourhood Cleaning Rule + RNM.
- Modelo SMOTE-ENN + RNM.
- Modelo Balanced Bagging Classifier.
- Modelo Easy Ensemble Classifier.

Notar que el detalle de los resultados de los modelos restantes se expone en el apartado A de los anexos.

4.4.2. Modelo caso base

En primera instancia, se lleva a cabo una validación cruzada de 5 iteraciones para ver los hiperparámetros óptimos mediante RandomizedSearchCV, en donde se evalúan 25 combinaciones dentro del espacio de hiperparámetros definido y se utiliza la métrica f1-score para medir su desempeño. Obteniendo la siguiente combinación:

Notar que todos los hiperparámetros excepto el número de capas ocultas, coincide con los valores asignados por defecto. Esta disminución en el número de capas ocultas simplifica el clasificador, lo que ayuda a evitar el sobreajuste del modelo.

Tabla 4.1: Hiperparámetros entregados por RandomizedSearchCV para el caso base.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Una vez están definidos lo hiperparámetros, se entrena la red neuronal y se evalúa su desempeño con el conjunto de evaluación definido anteriormente. A continuación se muestran los resultados obtenidos:

Tabla 4.2: Resultados de la evaluación del caso base.

	F1-score	Desviación estándar
Entrenamiento	0.9549	0.011
Testeo	0.6202	0.030

En la tabla 4.2 se puede ver que obtiene un f1-score cercano a 1.0 (valor máximo), indicando que el clasificador entrenado tiene una gran capacidad de predecir la etiqueta de los datos con los que fue entrenado. Por otro lado, al evaluar el mismo modelo en el conjunto de datos de evaluación, es decir, datos que nunca ha visto antes, el f1-score baja considerablemente. Esto indica que hay un problema de sobreajuste, ya que a pesar de que tiene buenos resultados durante el entrenamiento, al exponerlo con datos nuevos no presenta resultados del mismo nivel.

Además de lo anterior, la desviación estándar para el caso de testeo aumenta considerablemente, lo que se debe a que el f1-score obtenido al evaluar con el conjunto de evaluación depende mucho de como se etiquetaron los datos.

4.4.3. Modelo Random Over Sampler + RNM

En la figura 4.12 se puede visualizar gráficamente el resultado de aplicar el método de oversampling “Random Over Sampler”, en donde al simplemente duplicar los datos de la clase minoritaria no ayuda a disminuir la superposición entre estas.

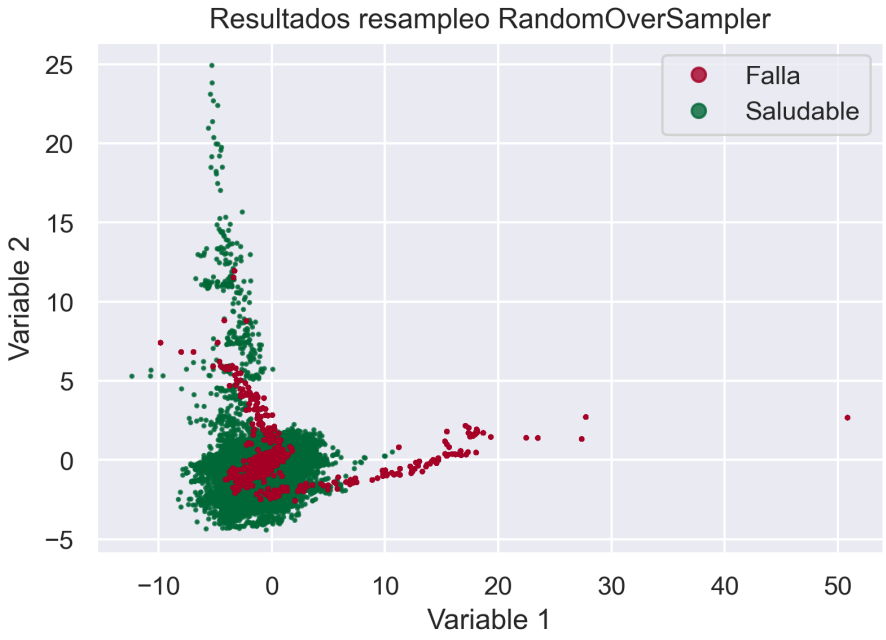

Figura 4.12: Resultados resampleo Random Over Sampler en dos dimensiones.

Cabe mencionar que la aplicación de este método resulta en una equilibración exacta entre la clase mayoritaria y minoritaria, es decir, se genera una cantidad de instancias de la clase minoritaria tal que ambas clases tengan la misma cantidad de datos, lo que se traduce en una razón de desbalance entre clases de 1:1.

Por otro lado, en la tabla 4.3 se aprecian los hiperparámetros obtenidos mediante el método de validación cruzada, destacando que el hiperparámetro alpha aumenta en comparación con su valor por defecto ($\alpha = 0.0001$), lo que alienta a tener pesos más pequeños, resultando en un límite de decisión menos curvado. En general, un alpha mayor indica que el modelo es menos sensible a datos ruidosos, lo que ayuda a combatir el overfitting [33].

Tabla 4.3: Hiperparámetros entregados por RandomizedSearchCV para modelo Random Over Sampler + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.01
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

En la tabla 4.4 se pueden apreciar el desempeño del modelo luego de ser entrenado, destacando que el simple hecho de duplicar instancias de la clase minoritaria puede mejorar considerablemente el desempeño en comparación con el caso base, lo que se traduce en un mayor f1-score de testeo.

Esto también se puede ver en la disminución de la desviación estándar de la métrica f1-score, mostrando que los modelos entrenados presentan una mayor estabilidad al cambiar la combinación de conjuntos de entrenamiento y evaluación.

Tabla 4.4: Resultados de la evaluación del modelo Random Over Sampler + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.947	0.009
Testeo	0.6720	0.019

Sumado a lo anterior, se obtiene un f1-score de entrenamiento similar al caso base, lo que demuestra que a pesar de que haya aumentado el desempeño en el testeo, aún hay señales de overfitting, ya que el modelo presenta un desempeño considerablemente mejor al ser evaluado en el conjunto de entrenamiento que en el de evaluación. No obstante, la aplicación del método de oversampling ha mostrado mitigar levemente el efecto de las clases desbalanceadas.

4.4.4. Modelo SMOTE + RNM

Con respecto al método de oversamplig "SMOTE", se puede observar el resultado del resampleo en la figura 4.13, en donde se ve claramente como se utiliza método de interpolación para generar instancias nuevas, lo que resalta fuertemente la naturaleza de la clase minoritaria. En específico, se utiliza el método de los K vecinos más cercanos para escoger los datos a interpolar, con $K = 5$, obteniendo así dos clases equilibradas exactamente.

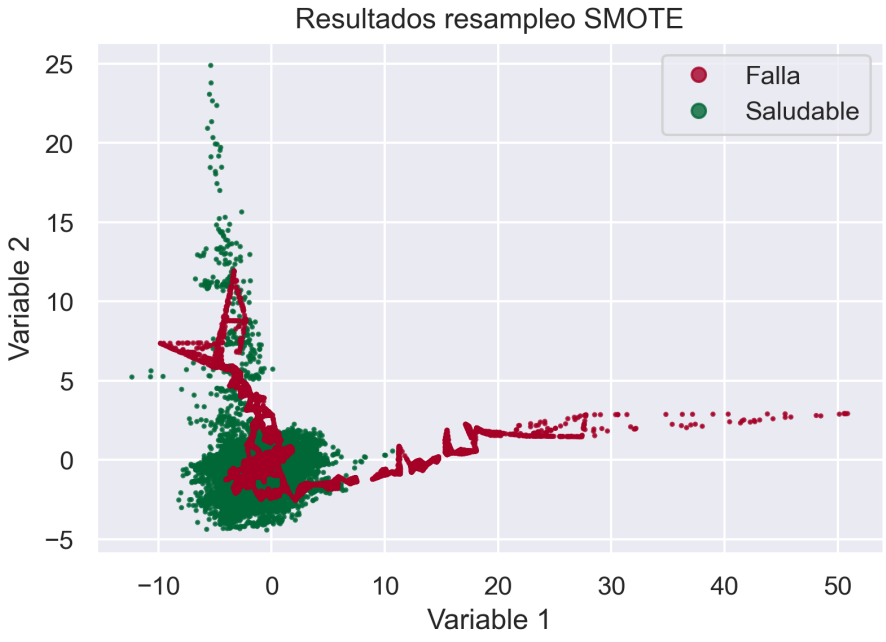

Figura 4.13: Resultados resampleo SMOTE en dos dimensiones.

En la tabla 4.5 se encuentran los hiperparámetros obtenidos, en donde se puede ver que son bastante similares con los casos anteriores.

Tabla 4.5: Hiperparámetros entregados por RandomizedSearchCV para modelo SMOTE + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Luego de entrenar el modelo se obtienen los resultados expuestos en la tabla 4.6, en la cual se puede ver que al incluir el método de oversampling mejora el rendimiento en comparación con el caso base. Cabe mencionar que a pesar de que el criterio para generar instancias nuevas en este método es más elaborado en contraste con el anterior, su desempeño es levemente menor, demostrando que un método más complejo no necesariamente es más eficiente.

Tabla 4.6: Resultados de la evaluación del modelo SMOTE + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9694	0.007
Testeo	0.6649	0.026

Notar también que se obtiene un f1-score de entrenamiento ligeramente mayor respecto a los casos anteriores, el cual consiste en un síntoma más agudo de overfitting. Además, la desviación estándar al momento de evaluar el modelo indica que no hay mucha estabilidad al momento de variar los conjuntos.

4.4.5. Modelo All-KNN + RNM

El primer método de resampleo tipo undersampling analizado es “All-KNN”, el cual consiste en reiterar múltiples veces el método ENN, mientras se va aumentando el número de vecinos más cercanos (K) en cada iteración. El resultado de la aplicación de este método se puede visualizar en la figura 4.14, la cual muestra que no hay una gran diferencia en comparación con el caso base. Esto se debe a que solo se eliminan unas pocas instancias cercanas al límite de decisión, sin importar la clase a la que pertenece. Debido a esto, no se obtiene un balanceo exacto entre clases, sino que se trabaja con una razón de desbalance de 1:48, ligeramente mejor el caso base (1:49).

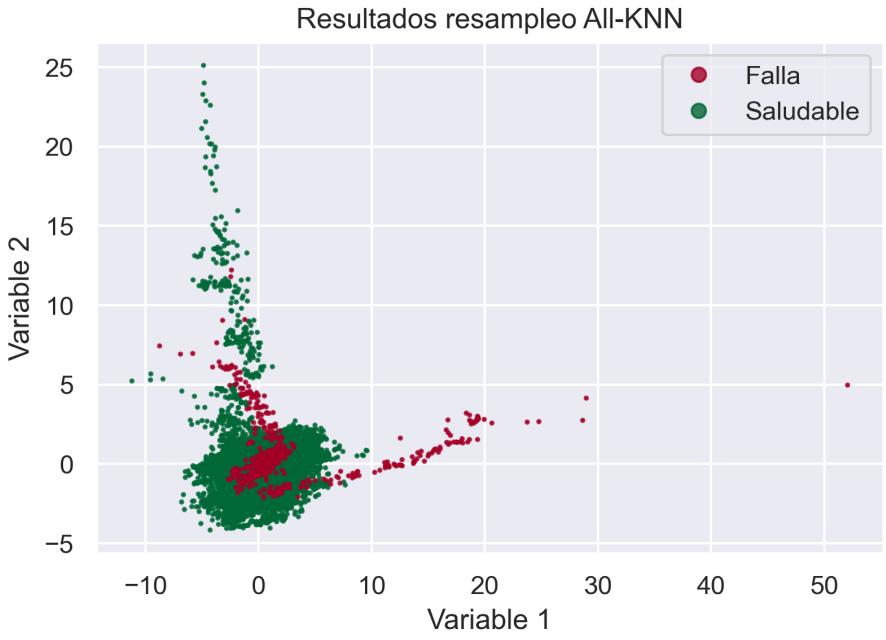

Figura 4.14: Resultados resampleo All-KNN en dos dimensiones.

Con respecto a los hiperparámetros expuestos en la tabla 4.7, son similares a los casos anteriores, destacando que se trabaja un alpha levemente mayor, lo que puede ayudar a combatir los efectos del overfitting.

Tabla 4.7: Hiperparámetros entregados por RandomizedSearchCV para modelo All-KNN + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.01
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Al evaluar el modelo múltiples veces, se obtienen los resultados presentes en la tabla 4.8, en donde se puede ver que a pesar de que se hayan eliminado pocos datos, mejora el rendimiento en contraste con el caso base. Esto muestra la importancia de la eliminación de instancias cercanas al límite de decisión, ya que ayuda al modelo de aprendizaje de máquinas a determinar un límite de decisión más eficiente.

No obstante lo anterior, el modelo presenta síntomas claros de overfitting, es específico, presenta un f1-score de entrenamiento considerablemente mayor al f1-score de testeo. Por otra parte, el modelo presenta una desviación estándar de su f1-score de testeo menor al caso base, pero similar a los otros modelos de resampleo.

Tabla 4.8: Resultados de la evaluación del modelo All-KNN + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9486	0.005
Testeo	0.6555	0.020

4.4.6. Modelo Neighbourhood Cleaning Rule + RNM

Al igual que en el caso anterior, el método "Neighbourhood Cleaning Rule" es una variación del método ENN, ya que primero elimina datos cercanos al límite de decisión mediante ENN y luego aplica el método CNN, el cual se encarga de eliminar datos lejanos al límite de decisión. Considerando esto, es de esperar que se eliminen pocas instancias, al igual que el caso anterior, resultando en que se tenga una razón de desbalance de 1:48.

En la figura 4.15, se puede ver claramente lo mencionado antes, en donde la distribución entre clases es similar al caso base.

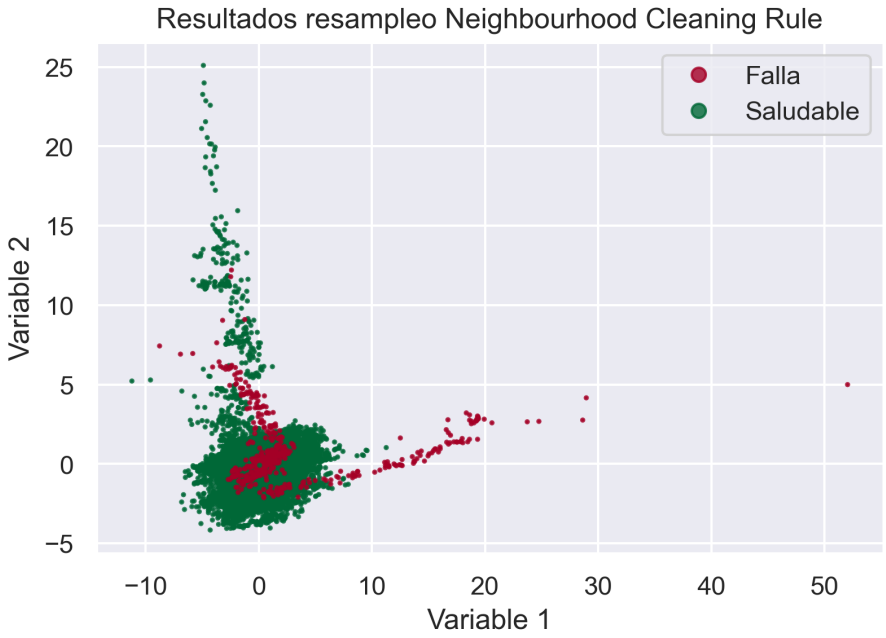

Figura 4.15: Resultados resampleo Neighbourhood Cleaning Rule en dos dimensiones.

Con respecto a los hiperparámetros y los resultados expuestos en las tablas 4.9 y 4.10 respectivamente, se tiene que son bastante similares al caso anterior. En donde nuevamente hay indicios de overfitting, además de que la eliminación de datos lejanos al límite de decisión no parece ayudar sustancialmente.

Tabla 4.9: Hiperparámetros entregados por RandomizedSearchCV para modelo Neighbourhood Cleaning Rule + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	30
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla 4.10: Resultados de la evaluación del modelo Neighbourhood Cleaning Rule + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9515	0.005
Testeo	0.6485	0.017

4.4.7. Modelo SMOTE-ENN + RNM

El último método de resampleo a analizar es "SMOTE-ENN", el cual combina la técnica de oversampling SMOTE con la técnica de undersampling ENN, aprovechando las cualidades de ambas. El resultado de esta combinación se puede apreciar en la figura 4.16, en la cual se aprecia que es bastante similar al gráfico de dispersión del método SMOTE analizado anteriormente, con la diferencia que en este caso se eliminan datos cercanos al límite de decisión después de aplicar el método de oversampling.

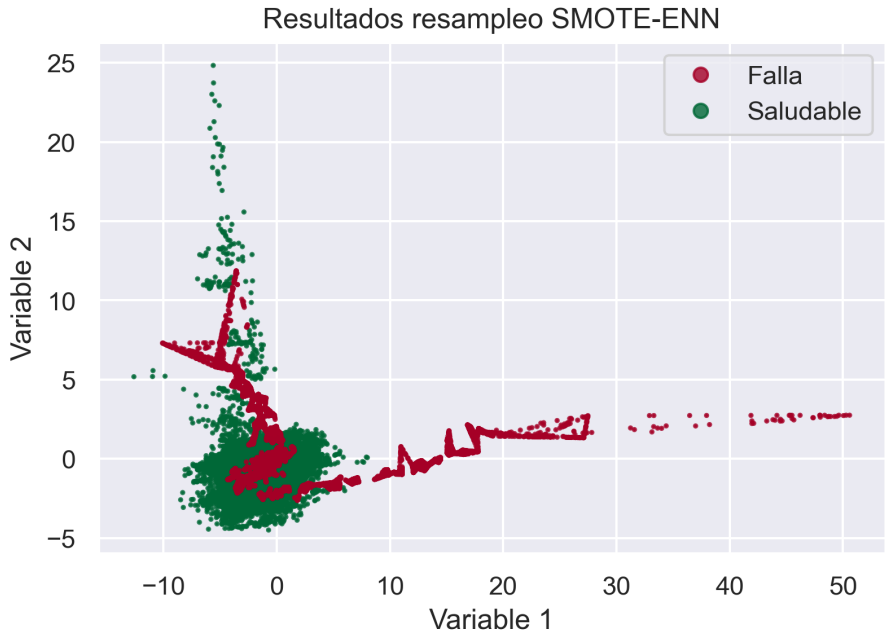

Figura 4.16: Resultados resampleo SMOTE-ENN en dos dimensiones.

Sumado a lo anterior, al aplicar este método de resampleo se obtiene una razón de desbalance cercano a 1:1, en donde hay un poco más datos de fallas que saludables. Esto que se debe a que el método de oversampling realiza un balance exacto y luego el método de undersampling elimina más datos de la clase saludable que de la clase de fallas.

Tabla 4.11: Hiperparámetros entregados por RandomizedSearchCV para modelo SMOTE-ENN + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla 4.12: Resultados de la evaluación del modelo SMOTE-ENN + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9491	0.006
Testeo	0.6809	0.015

Notar que los hiperparámetros de la tabla 4.11 son iguales a los del método SMOTE, por lo que lo único que diferencia estos métodos es la posterior aplicación de un método de undersampling por parte de SMOTE-ENN. Esta diferencia se traduce en una mejora en el desempeño, lo que se puede apreciar en el f1-score de testeo de la tabla 4.12, el cual es el más alto de todos los modelos evaluados. Además, presenta una desviación estándar del f1-score de testeo baja en comparación con los otros modelos, demostrando su estabilidad al variar los conjuntos de entrenamiento y evaluación.

No obstante lo anterior, este caso también presenta síntomas de overfitting, comprometiendo su capacidad de generalización, o en otras palabras, su capacidad de detectar fallas nuevas.

4.4.8. Modelo Balanced Bagging Classifier

Se implementa un algoritmo tipo bagging consistido por un conjunto de cinco árboles de decisión, entrenados separadamente por subconjuntos de la base de datos resampleados mediante el método de undersampling Tomek links.

Cabe mencionar que se escoge este método de resampleo debido a que es el que presenta los mejores resultados al ser evaluado manualmente, al igual que la cantidad de árboles de decisión utilizados. No se estima necesario la aplicación de herramientas como RandomizedSearchCV, esto debido a la escasa cantidad hiperparámetros necesarios a ajustar.

En la tabla 4.13 se exponen los resultados del modelo, destacando el alto nivel del f1-score de entrenamiento, lo que es una señal clara de overfitting, ya que clasifica de forma correcta práctica-

Tabla 4.13: Resultados de la evaluación del modelo Balanced Bagging Classifier.

	F1-score	Desviación estándar
Entrenamiento	0.9953	0.002
Testeo	0.5869	0.119

mente todos los datos con los que fue entrenado cada vez. Sumado a que el f1-score de testeo es considerablemente menor, incluso menor que en el caso base, por lo que en este caso, esta técnica no fue de ayuda para solucionar el problema de clases desbalanceadas.

Adicionalmente, la desviación estándar del f1-score de testeo es significativamente mayor a los otros casos, lo que muestra lo inestable que es el modelo al variar los conjuntos de entrenamiento y evaluación, es decir, hay casos en que se desempeña muy bien, pero otros en que se desempeña muy mal.

4.4.9. Modelo Easy Ensemble Classifier

Este modelo consiste en un ensamblaje de tipo bagging, en donde cada clasificador débil es un algoritmo de tipo boosting llamado Adaboost. Cada uno de estos clasificadores es entrenado con un subconjunto de la base de datos, el cual es resampleado por el método de undersampling "Random Under Sampling" (RUS).

Con respecto a los hiperparámetros, este modelo está compuesto por 60 clasificadores Adaboost, en donde cada uno de ellos utiliza 30 árboles de decisión para construir su algoritmo tipo boosting. Al igual que en el modelo anterior, se ajustan los hiperparámetros de manera manual.

Tabla 4.14: Resultados de la evaluación del modelo Easy Ensemble Classifier.

	F1-score	Desviación estándar
Entrenamiento	0.6161	0.012
Testeo	0.4273	0.032

A pesar de la gran complejidad del modelo, este no logra aprender la naturaleza subyacente de los datos, lo que se puede ver claramente en el bajo f1-score de entrenamiento y de testeo, expuestos en la tabla 4.14. Al contrario que los casos anteriores, este modelo sufre de underfitting, es decir, no es capaz de capturar los patrones subyacentes en los datos.

Una de las posibles razones de esto es tener una base de datos no representativa del problema, en este caso en particular, la implementación del método de resampleo RUS genera algo similar, ya que la eliminación de muchas instancias de manera aleatoria desecha información importante para representar el problema. Esto se puede ver en los resultados del modelo "Random Under Sampling + RNM", expuestos en la tabla A.20, en donde se tienen resultados similares a este caso.

4.5. Evaluación capacidad de generalización

Con el fin de discernir la utilidad real de los modelos entrenados, se vuelve necesario evaluarlos con datos nuevos, para así ver si realmente son capaces de detectar fallas nuevas en los equipos. Para realizar esto, se escoge solo el modelo con mejor desempeño mostrado durante la etapa de entrenamiento, esto debido a la gran cantidad de modelos.

Considerando el desempeño de los modelos entrenados en este estudio, se decide que se evaluará el modelo SMOTE-ENN + RNM, el cual obtuvo un f1-score de testeo promedio de 0.68 aproximadamente. En específico, se utiliza el modelo con mejor desempeño en todas las iteraciones en las que fue evaluado, el cual presenta un f1-score de 0.71.

Para entender de manera completa las capacidades y limitaciones del modelo escogido, se pone a prueba tanto la capacidad de detectar la falla con la que fue entrenado, como una falla nueva de otro equipo.

4.5.1. Detección de falla de entrenamiento

Para ver la capacidad de detección de la falla con la que fue entrenado, se le pide al modelo diagnosticar la base de datos sin separarla en subconjuntos de entrenamiento y testeo, lo que se puede apreciar en la figura 4.17. En este gráfico, se representa la falla mediante las vibraciones de dirección X del descanso del motor, es específico el lado del acople. En donde los puntos negros corresponden a datos clasificados como saludable, las cruces rojas son datos clasificados como falla y la línea verde es una representación visual de cuando ocurre la falla.

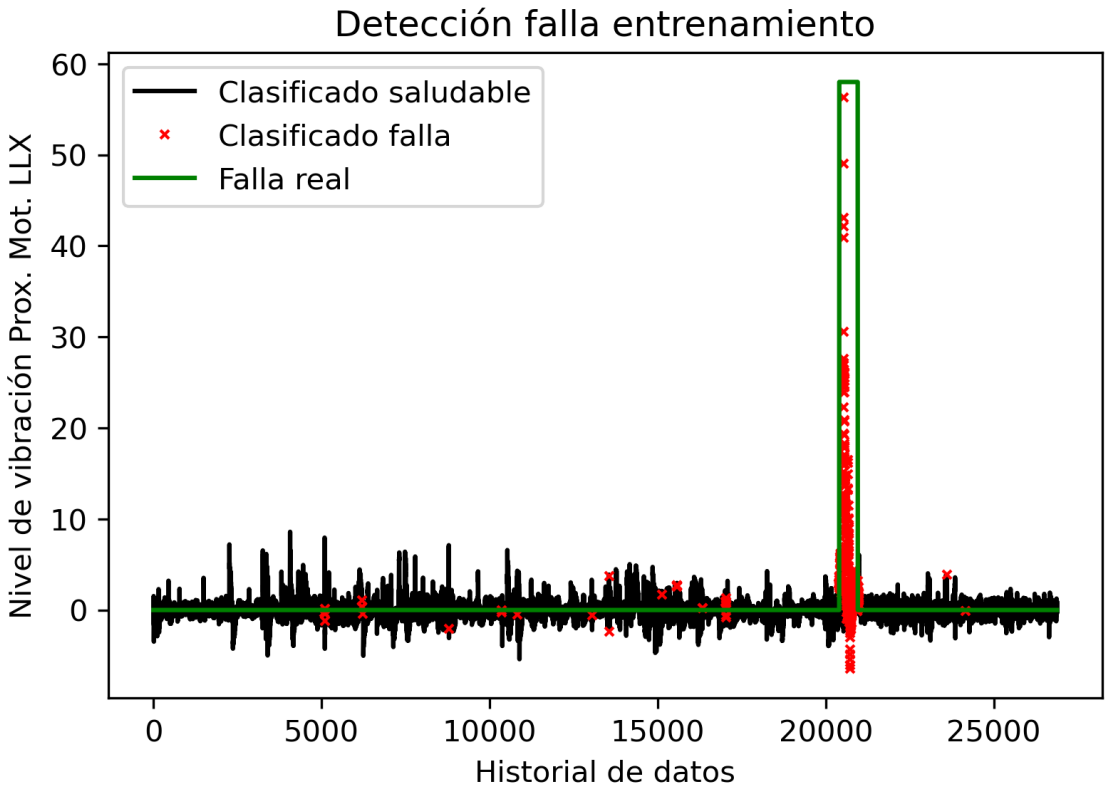

Figura 4.17: Visualización detección falla de entrenamiento.

En la figura anterior se aprecia como el modelo es capaz de detectar la falla de entrenamiento sin problemas, no obstante, el modelo también clasifica erróneamente como falla a datos que en realidad representan un equipo con un estado saludable, los cuales se pueden ver repartidos a lo largo del historial de monitoreo y no caen dentro del escalón verde.

4.5.2. Detección de falla nueva

Para la evaluación de la capacidad de generalización del modelo escogido se utiliza un historial de monitoreo de otra bomba hidráulica igual, el cual incluye información sobre una falla caracterizada por un alto nivel de vibraciones, al igual que la falla de entrenamiento. La falla en cuestión es representada en la figura 4.18, en donde se utiliza la variable de las vibraciones de dirección X del descanso del acople de la bomba.

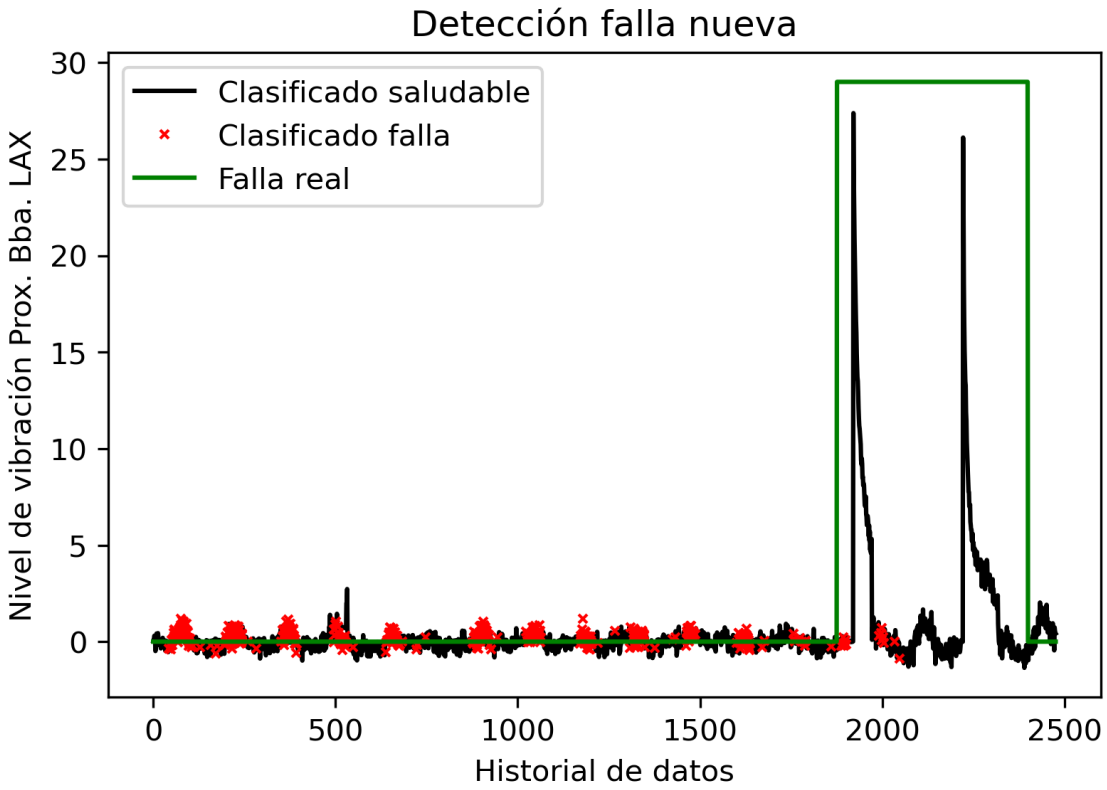

Figura 4.18: Visualización detección falla de entrenamiento.

En el gráfico se aprecia que el modelo no realiza un buen trabajo en la clasificación de los estados de salud del equipo, ya que detecta muchas fallas cuando la máquina está saludable y no detecta de manera apropiada la falla real, solo algunos puntos al inicio de esta. Este resultado indica que el modelo entrenado tiene una mala capacidad de generalización, ya que no es capaz de detectar de manera correcta fallas nuevas.

A pesar de realizar un buen trabajo en detectar la falla con que fue entrenado, el modelo presenta un desempeño deficiente al ser expuesto a datos nuevos, lo que se puede deber a que el modelo sufre de overfitting. Esto no solo se puede ver al evaluar su desempeño con fallas nuevas, sino que se ve también en las métricas obtenidas al durante el entrenamiento, ya que presenta un f1-score notoriamente mayor al f1-score de testeo, ver tabla 4.12.

Otro factor importante que puede explicar la baja capacidad de generalización expuesta por el modelo es que las variables físicas de cada bomba suelen presentar un comportamiento diferente. Esto se debe a que cualquier mínima variación en la instalación del equipo puede variar significativamente en la medición de los sensores, lo que dificulta el desarrollo de un modelo con buena capacidad de generalización.

La diferencia en el comportamiento de los datos recopilados por los sensores también puede producir cuando un equipo es intervenido para realizar una mantención o arreglo. Debido a lo

anterior, en algunos casos se puede volver necesario reentrenar el modelo de decisión. Sin embargo, si se considera la escasez de datos de fallas en este tipo de problemas, se vuelve un obstáculo tener que reentrenar reiteradamente un modelo para los equipos a monitorear.

Además de lo anterior, la capacidad limitada de generalización mostrada por el modelo se puede deber a que las fallas de entrenamiento y evaluación pueden tener naturalezas muy distintas. Por ejemplo, para este caso, la falla de entrenamiento está caracterizada por vibraciones en el motor, por lo que es de esperar que el modelo tenga dificultades para detectar una falla caracterizada por vibraciones en la bomba.

Otra razón de los resultados obtenidos puede ser la dificultad de definir una etiqueta de fallas precisa para entrenar los modelos, lo que en este caso se debe a la cantidad de información limitada del estado de salud de los equipos a lo largo del tiempo. Esto puede generar que el modelo aprenda patrones erróneos, traduciéndose en un desempeño y una capacidad de generalización pobre.

Si se quisiera implementar este algoritmo para todos los equipos de las estaciones de bomba, se recomienda el entrenamiento de un modelo para cada equipo a monitorear. Sin embargo, la dificultad para definir las etiquetas y el reiterado cambio en el comportamiento de las variables sería un gran obstáculo, ya que implicaría los modelos se vean constantemente obsoletos y que su desempeño se vea en cierto grado limitado por la naturaleza de la falla de entrenamiento.

Capítulo 5

Conclusiones y trabajos futuros

En este estudio se presenta la implementación y evaluación de dos enfoques que buscan solucionar el problema de clases desbalanceadas en aplicaciones de la industria. Ambos enfoques utilizan algoritmos de aprendizaje de máquinas supervisados, con la particularidad que el primer enfoque combina métodos de resampleo y redes neuronales, mientras que el segundo combina métodos de resampleo y métodos de ensamblaje.

Luego de preprocesar exitosamente los datos y entrenar los modelos, se obtiene que, en la mayoría de los casos, el primer enfoque sirve para mejorar el desempeño con respecto al caso base. En el caso específico de los métodos de oversampling, los que presentan mejor desempeño son aquellos con algoritmos internos menos complejos, destacando ROS y SMOTE. Mientras que los métodos de undersampling con mejor desempeño son aquellos enfocados en la eliminación de datos cercanos al límite de decisión, destacando All-KNN y Neighbourhood Cleaning Rule. Por otro lado, al combinar las cualidades de cada técnica, se obtienen los mejores resultados, presentados por SMOTE-ENN.

Con respecto al segundo enfoque, este presenta resultados inferiores en comparación con el caso base, lo que principalmente se debe a la naturaleza del método de resampleo utilizado (RUS), ya que a pesar de presentar el beneficio de tener un bajo costo computacional, este presenta resultados deficientes en todos los escenarios que es aplicado.

Al evaluar la capacidad de generalización del modelo escogido (SMOTE-ENN + RNM), se obtiene que esta es baja. Lo que principalmente se debe a que el modelo sufre de overfitting, ya que tiene una gran capacidad para detectar la falla con la que fue entrenado, pero una capacidad limitada para detectar fallas nuevas. La limitación mostrada por el modelo se puede deber a la diferencia del comportamiento de los datos de distintas bombas, la diferencia en la naturaleza de la falla de entrenamiento y evaluación, y la dificultad de definir las etiquetas de los datos de manera precisa.

Finalmente, es posible afirmar que se logra desarrollar un modelo de diagnóstico de fallas ba-

sado en aprendizaje de máquinas que presenta una mejora en el desempeño al trabajar con datos desbalanceados, pero que simultáneamente presenta una capacidad de generalización limitada. Considerando lo anterior, se recomienda el desarrollo de un modelo por cada equipo a monitorear.

5.1. Trabajos futuros

Debido a la capacidad de generalización limitada del modelo desarrollado, los futuros esfuerzos deben estar enfocados en mejorar aquello. Para lograr una mejora en esto se pueden tomar diversas direcciones, tal como ajustar nuevamente los hiperparámetros de las redes neuronales para simplificar el modelo, utilizar otro tipo de clasificador menos propenso a sufrir de overfitting, entre otros.

Además de lo anterior, la escasez de datos de fallas disponibles para el entrenamiento de modelos y el constante cambio en el comportamiento de los datos en los equipos, consisten en un gran obstáculo para desarrollar modelos basados en aprendizaje supervisado con buena capacidad de generalización. Considerando lo anterior, se abre la posibilidad de explorar otras alternativas, tal como la implementación de técnicas de aprendizaje no supervisado, por ejemplo, algoritmos de detección de anomalías o algoritmos de detección de novedades.

Bibliografía

[1] Lei, Y., Yang, B., Jiang, X., Jia, F., Li, N., & Nandi, A. K. (2020). *Applications of machine learning to machine fault diagnosis: A review and roadmap. Mechanical Systems and Signal Processing*, 138(106587), 106587. <https://doi.org/10.1016/j.ymssp.2019.106587>

[2] Alpaydin, E. (2020). *Introduction to Machine Learning (4a ed.)*, MIT Press.

[3] SUN, Y., WONG, A. K. C. & KAMEL, M. S. (2009). *Classification of imbalanced data: a review. International Journal of Pattern Recognition and Artificial Intelligence*, 23(04), 687-719. <https://doi.org/10.1142/s0218001409007326>

[4] Stefanowski, J. (2015). *Dealing with Data Difficulty Factors While Learning from Imbalanced Data. Studies in Computational*, 23(04), 333-363. <https://doi.org/10.1007/978-3-319-18781-5_17>

[5] Al-Rahman Al-Serw, N. (2021, 31 diciembre). *Undersampling and oversampling: An old and a new approach. Medium.*, <https://medium.com/analytics-vidhya/undersampling-and-oversampling-an-old-and-a-new-approach-4f984a0e8392>

[6] Batista, G. E. A. P. A., Prati, R. C. & Monard, M. C. (2004). *A study of the behavior of several methods for balancing machine learning training data.*, ACM SIGKDD Explorations Newsletter, 6(1), 20-29. <https://doi.org/10.1145/1007730.1007735>

[7] Haibo He, Yang Bai, Garcia, E. A. & Shutao Li. (2008). *ADASYN: Adaptive synthetic sampling approach for imbalanced learning.*, IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence). <https://doi.org/10.1109/ijcnn.2008.4633969>

[8] Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002). *SMOTE: Synthetic Minority Over-sampling Technique.*, Journal of Artificial Intelligence Research, 16, 321-357. <https://doi.org/10.1613/jair.953>

[9] Upadhyay, Kamlesh & Kaur, Prabhjot & Prasad, Svav. (2021). *State of the Art on Data level methods to address Class Imbalance Problem in Binary Classification.*, 8. 975-903. <https://www.researchgate.net/publication/350132780>

[10] Han, H., Wang, W. Y. & Mao, B. H. (2005). *Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning.*, Lecture Notes in Computer Science, 878-887. <https://doi.org/10.1007/11538059_91>

[11] Nguyen, H. M., Cooper, E. W. & Kamei, K. (2011). *Borderline over-sampling for imbalanced data classification.*, International Journal of Knowledge Engineering and Soft Data Paradigms, 3(1), 4. <https://doi.org/10.1504/ijkesdp.2011.039875>

[12] I Tomek. (1976). *Two Modifications of CNN.*, IEEE Transactions on Systems, Man, and Cybernetics, SMC–6(11), 769-772. <https://doi.org/10.1109/tsmc.1976.4309452>

[13] Wilson, D. L. (1972). *Asymptotic Properties of Nearest Neighbor Rules Using Edited Data.*, IEEE Transactions on Systems, Man, and Cybernetics, SMC–2(3), 408-421. <https://doi.org/10.1109/tsmc.1972.4309137>

[14] Ivan Tomek. (1976). *An Experiment with the Edited Nearest-Neighbor Rule.*, IEEE Transactions on Systems, Man, and Cybernetics, SMC–6(6), 448-452. <https://doi.org/10.1109/tsmc.1976.4309523>

[15] Hart, P. (1968). *The condensed nearest neighbor rule (Corresp.).*, IEEE Transactions on Information Theory, 14(3), 515-516. <https://doi.org/10.1109/tit.1968.1054155>

[16] *3. Under-sampling — Version 0.10.0.*, (s.f.). <https://imbalanced-learn.org/stable/under_sampling.html>

[17] Laurikkala, J. (2001). *Improving Identification of Difficult Small Classes by Balancing Class Distribution.*, Artificial Intelligence in Medicine, 63-66. <https://doi.org/10.1007/3-540-48229-6_9>

[18] Kubat, M. & Matwin, S. (1996). *Addressing the Curse of Imbalanced Training Sets: One-Sided Selection.*, International Conference on Machine Learning, 179-186. <https://sci2s.ugr.es/keel/pdf/algorithm/congreso/kubat97addressing.pdf>

[19] Zhang, J. & Mani, I. (2003). *KNN approach to unbalanced data distributions: a case study involving information extraction.*, In Proceedings of workshop on learning from imbalanced datasets, volume 126. 2003. <https://www.site.uottawa.ca/~nat/Workshop2003/jzhang.pdf>

[20] Krawczyk, B. (2016). *Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence.*, 5(4), 221-232. <https://doi.org/10.1007/s13748-016-0094-0>

[21] Rocca, J. R. (2019, 22 abril). *Ensemble methods: Bagging, boosting and stacking.*, <https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205>

[22] Richard Maclin & David W. Opitz. (1997). *An empirical evaluation of bagging and boosting.*, National Conference on Artificial Intelligence, 546-551.

[23] Wang, S. & Yao, X. (2009). *Diversity analysis on imbalanced data sets by using ensemble models.*, 2009 IEEE Symposium on Computational Intelligence and Data Mining. <https://doi.org/10.1109/cidm.2009.4938667>

[24] Chen, Chao & Breiman, Leo. (2004). *Using Random Forest to Learn Imbalanced Data.*, University of California, Berkeley.

[25] Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J. & Napolitano, A. (2010). *RUSBoost: A Hybrid Approach to Alleviating Class Imbalance.*, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans. 40(1), 185-197. <https://doi.org/10.1109/tsmca.2009.2029559>

[26] *sklearn.ensemble.AdaBoostClassifier.*, (s. f.). scikit-learn. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>

[27] Xu-Ying Liu, Jianxin Wu & Zhi-Hua Zhou. (2009b). *Exploratory Undersampling for Class-Imbalance Learning.*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). 39(2), 539-550. <https://doi.org/10.1109/tsmcb.2008.2007853>

[28] *1.17. Neural network models (supervised).*, (s. f.). scikit-learn. <https://scikit-learn.org/stable/modules/neural_networks_supervised.html>

[29] Mate Labs. (2017, 1 noviembre). *Everything you need to know about Neural Networks.*, HackerNoon. <https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491>

[30] *sklearn.neuralnetwork.MLPClassifier.*, (s.f.). scikit-learn. <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html>

[31] *Documentación librería imbalanced-learn - Versión 0.9.1.*, (s.f.). <https://imbalanced-learn.org/stable/>

[32] *sklearn.modelselection.RandomizedSearchCV.*, (s.f.). scikit-learn. <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html>

[33] *Varying regularization in Multi-layer Perceptron.*, (s.f.). scikit-learn. <https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html>

Anexo A

Resultados modelos restantes

A.1. Modelo ADASYN + RNM

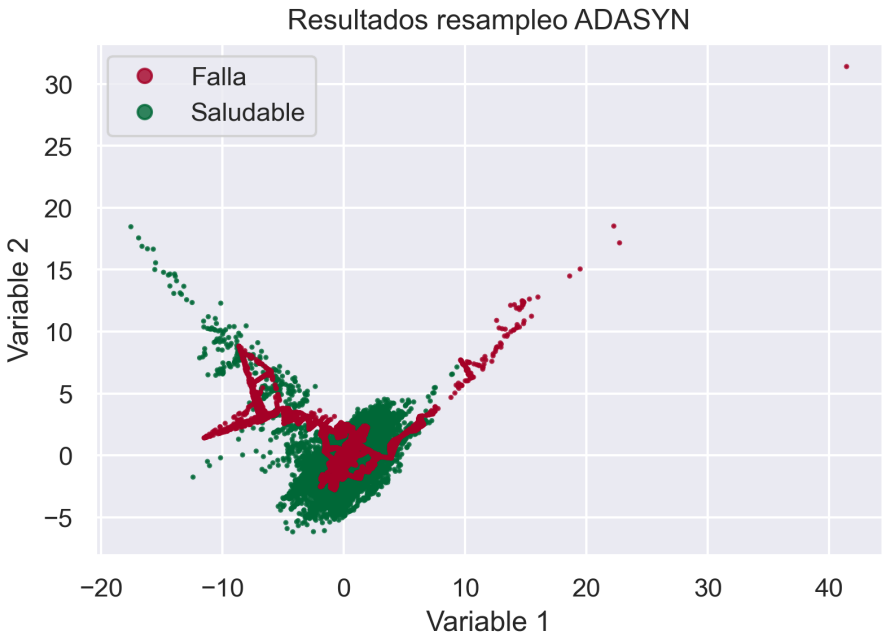

Figura A.1: Resultados resampleo ADASYN en dos dimensiones.

Tabla A.1: Hiperparámetros entregados por RandomizedSearchCV para modelo ADASYN + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.001
Función de activación	Tanh
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla A.2: Resultados de la evaluación del modelo ADASYN + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9739	0.004
Testeo	0.5348	0.041

A.2. Modelo Borderline SMOTE + RNM

Se utiliza la versión borderline-1, la cual utiliza los datos que al menos la mitad de sus vecinos más cercanos son de la misma clase.

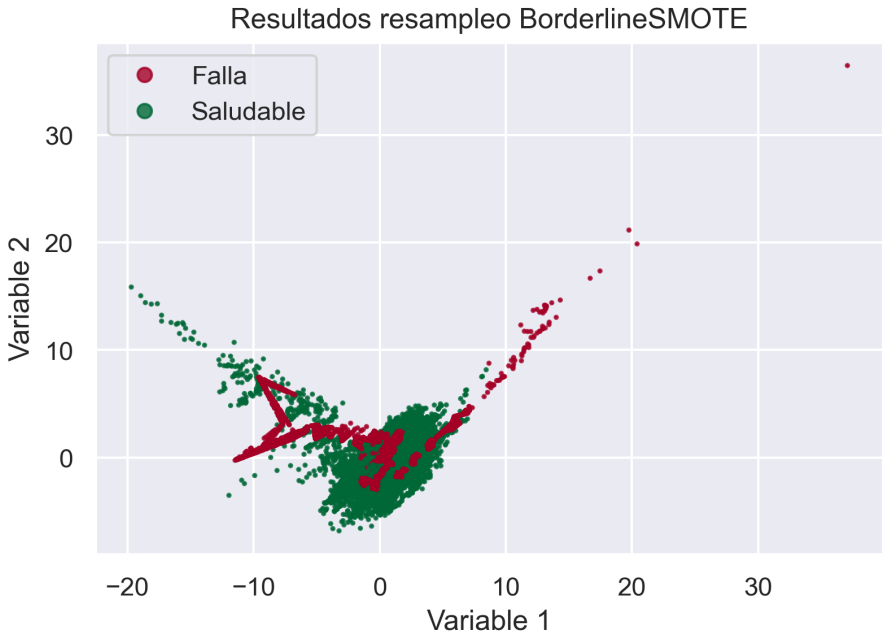

Figura A.2: Resultados resampleo Borderline SMOTE en dos dimensiones.

Tabla A.3: Hiperparámetros entregados por RandomizedSearchCV para modelo BorderlineSMOTE + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	20
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla A.4: Resultados de la evaluación del modelo BorderlineSMOTE + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9398	0.010
Testeo	0.5624	0.016

A.3. Modelo SVMSMOTE + RNM

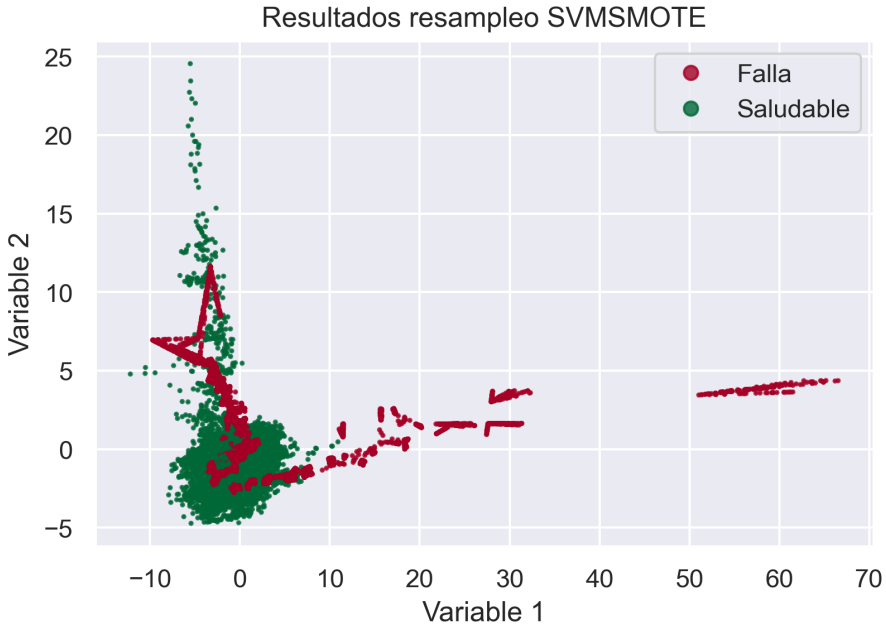

Figura A.3: Resultados resampleo SVMSMOTE en dos dimensiones.

Tabla A.5: Hiperparámetros entregados por RandomizedSearchCV para modelo SVMSMOTE + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	30
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla A.6: Resultados de la evaluación del modelo SVMSMOTE + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9475	0.009
Testeo	0.6442	0.014

A.4. Modelo Cluster Centroids + RNM

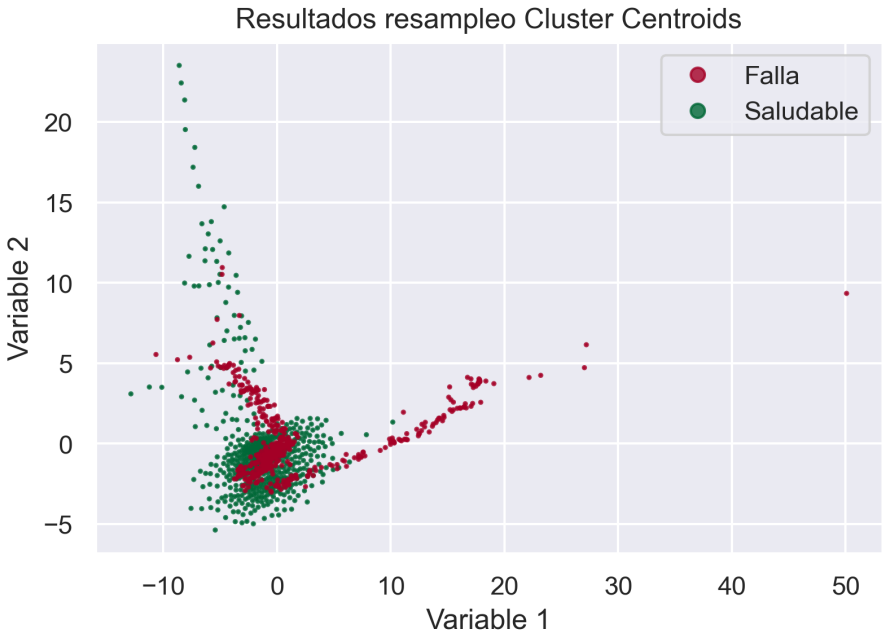

Figura A.4: Resultados resampleo Cluster Centroids en dos dimensiones.

Tabla A.7: Hiperparámetros entregados por RandomizedSearchCV para modelo Cluster Centroids + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	25
Alpha	0.01
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	1000

Tabla A.8: Resultados de la evaluación del modelo Cluster Centroids + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.7656	0.012
Testeo	0.5416	0.014

A.5. Modelo Condensed Nearest Neighbour + RNM

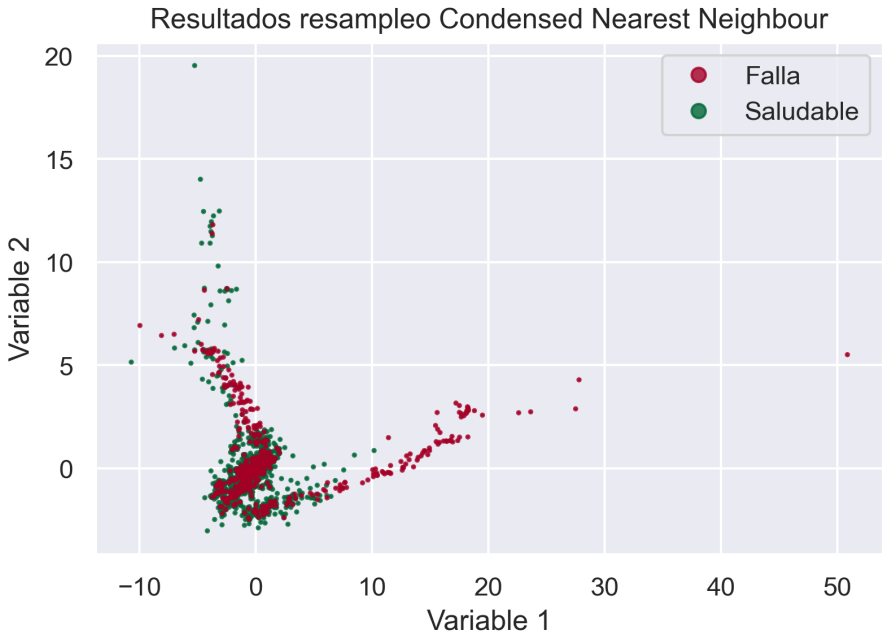

Figura A.5: Resultados resampleo Condensed Nearest Neighbour en dos dimensiones.

Tabla A.9: Hiperparámetros entregados por RandomizedSearchCV para modelo Condensed Nearest Neighbour + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	1000

Tabla A.10: Resultados de la evaluación del modelo Condensed Nearest Neighbour + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.8451	0.025
Testeo	0.5370	0.033

A.6. Modelo Edited Nearest Neighbours + RNM

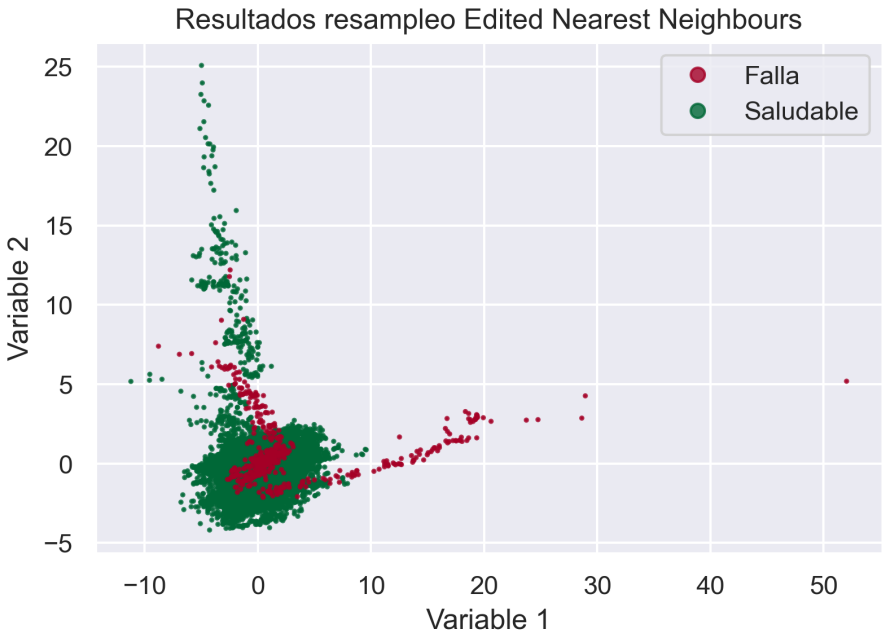

Figura A.6: Resultados resampleo Edited Nearest Neighbours en dos dimensiones.

Tabla A.11: Hiperparámetros entregados por RandomizedSearchCV para modelo Edited Nearest Neighbours + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	30
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla A.12: Resultados de la evaluación del modelo Edited Nearest Neighbours + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9558	0.005
Testeo	0.6349	0.024

A.7. Modelo Repeated Edited Nearest Neighbours + RNM

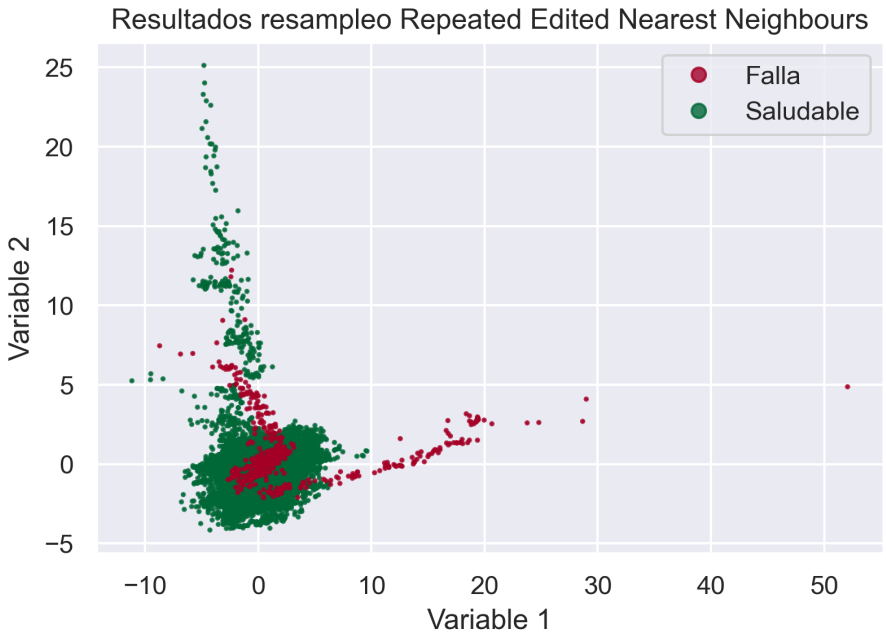

Figura A.7: Resultados resampleo Repeated Edited Nearest Neighbours en dos dimensiones.

Tabla A.13: Hiperparámetros entregados por RandomizedSearchCV para modelo Repeated Edited Nearest Neighbours + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	30
Alpha	0.001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla A.14: Resultados de la evaluación del modelo Repeated Edited Nearest Neighbours + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9550	0.004
Testeo	0.6432	0.034

A.8. Modelo Near Miss + RNM

Se utiliza la variación Near Miss 1, la cual selecciona las muestras de la clase mayoritaria que tienen la menor distancia promedio a las tres muestras más cercanas de la clase minoritaria. Para mejor visualización de ambas clases, se aumenta la transparencia de los puntos.

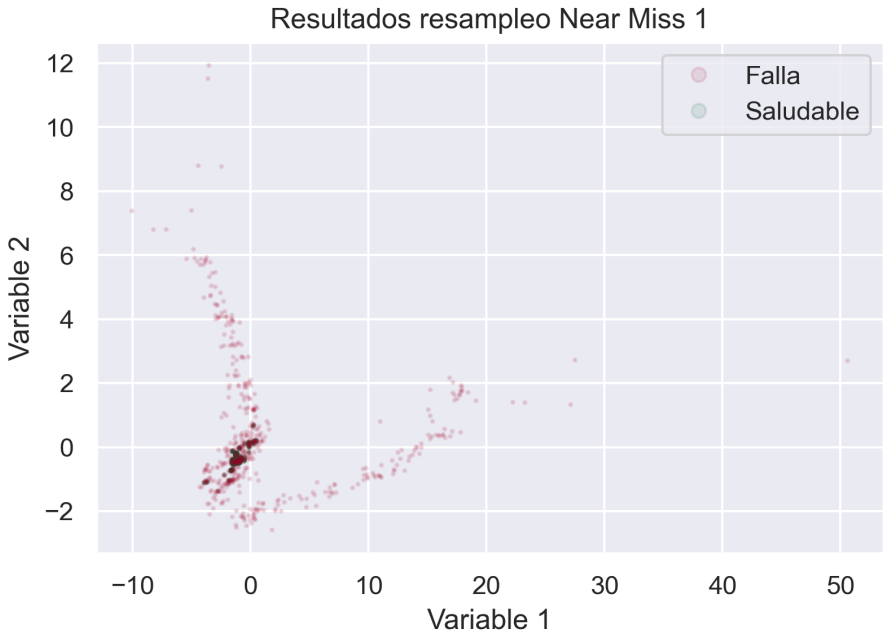

Figura A.8: Resultados resampleo Near Miss en dos dimensiones.

Tabla A.15: Hiperparámetros entregados por RandomizedSearchCV para modelo Near Miss + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	40
Alpha	0.001
Función de activación	Relu
Parada temprana	True
Número máximo de iteraciones	1000

Tabla A.16: Resultados de la evaluación del modelo Near Miss + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.0857	0.007
Testeo	0.0676	0.002

A.9. Modelo One Sided Selection + RNM

Figura A.9: Resultados resampleo One Sided Selection en dos dimensiones.

Tabla A.17: Hiperparámetros entregados por RandomizedSearchCV para modelo One Sided Selection + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.01
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla A.18: Resultados de la evaluación del modelo One Sided Selection + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9498	0.010
Testeo	0.6453	0.023

A.10. Modelo Random Under Sampler + RNM

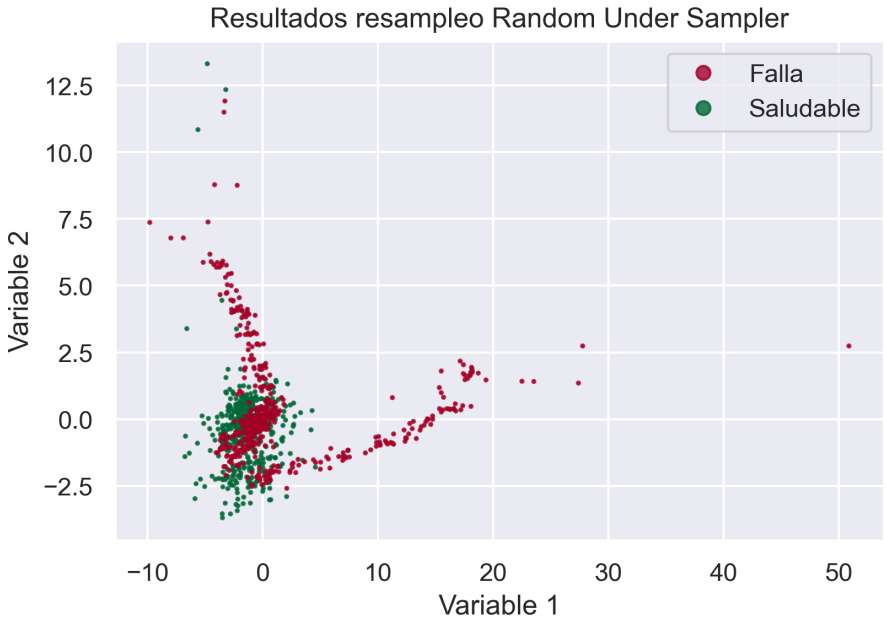

Figura A.10: Resultados resampleo Random Under Sampler en dos dimensiones.

Tabla A.19: Hiperparámetros entregados por RandomizedSearchCV para modelo Random Under Sampler + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	30
Alpha	0.01
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	1000

Tabla A.20: Resultados de la evaluación del modelo Random Under Sampler + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.6228	0.034
Testeo	0.4129	0.029

A.11. Modelo Tomek Links + RNM

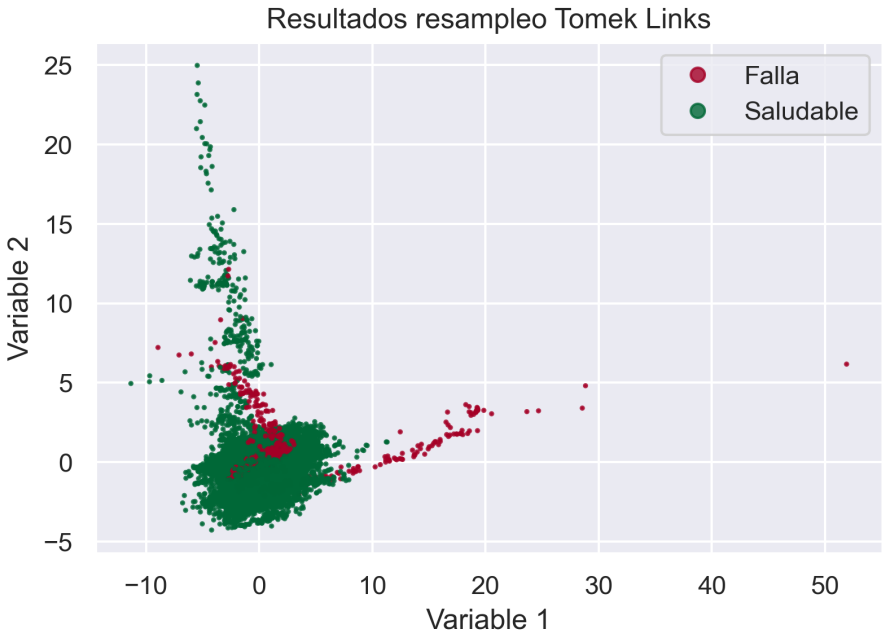

Figura A.11: Resultados resampleo Tomek Links en dos dimensiones.

Tabla A.21: Hiperparámetros entregados por RandomizedSearchCV para modelo Tomek Links + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.01
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla A.22: Resultados de la evaluación del modelo Tomek Links + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9458	0.008
Testeo	0.6351	0.019

A.12. Modelo SMOTE + Tomek Links + RNM

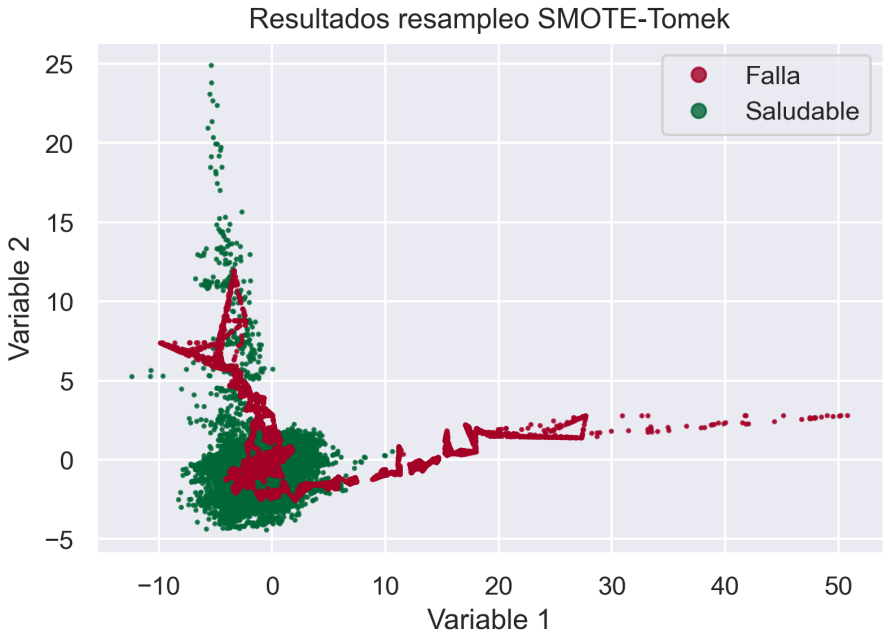

Figura A.12: Resultados resampleo SMOTE-Tomek Links en dos dimensiones.

Tabla A.23: Hiperparámetros entregados por RandomizedSearchCV para modelo SMOTE-Tomek + RNM.

	Hiperparámetros óptimos
Número de capas ocultas	35
Alpha	0.0001
Función de activación	Relu
Parada temprana	Falso
Número máximo de iteraciones	200

Tabla A.24: Resultados de la evaluación del modelo SMOTE-Tomek + RNM.

	F1-score	Desviación estándar
Entrenamiento	0.9719	0.006
Testeo	0.6602	0.029

A.13. Modelo Balanced Random Forest Classifier

Tabla A.25: Resultados de la evaluación del modelo Balanced Random Forest Classifier.

	F1-score	Desviación estándar
Entrenamiento	0.7720	0.012
Testeo	0.4584	0.019

A.14. Modelo RUS Boost Classifier

Tabla A.26: Resultados de la evaluación del modelo RUS Boost Classifier.

	F1-score	Desviación estándar
Entrenamiento	0.6189	0.050
Testeo	0.2219	0.103

Anexo B

Códigos

B.1. Gráficos de dispersión

Código B.1: Script para generar gráficos de dispersión.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from imblearn.over_sampling import *
from imblearn.under_sampling import *
from imblearn.combine import SMOTEENN, SMOTETomek

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import seaborn as sns

import os
import sys
PROJECT_ROOT = os.path.abspath(os.path.join(
              os.path.dirname(__file__),
              os.pardir)
)
sys.path.append(PROJECT_ROOT)
import utils

#Cargar datos falla entrenamiento
data = pd.read_csv('C:/Users/raimu/Desktop/Memoria/data/Bba118.csv')
data = data.set_index('Unnamed: 0')

#Cargar datos falla evaluación
df_111_v2=pd.read_pickle('C:/Users/raimu/Desktop/Memoria/data/data_clean/
    ↪ data_Bomba Impulsión 111 EB1.pkl')
data = df_111_v2

# IMPORTANTE: En este punto del código se lleva a cabo el preprocesamiento de
    ↪ los datos, pero debido a que incluye material proporcionado por una empresa,
    ↪ no se puede incluir.

#Definir posición inicio y fin etiquetas respectivamente
```

```
label_i=20400
label_f=20951

#Definir vector de etiquetas
y=np.zeros((len(data_processed),1))
y[label_i:label_f]=1

# Transformar a numpy array para poder aplica scaler
data_processed_pd = data_processed
data_processed = data_processed.to_numpy()

# Estandarización datos/Aplicación scaler
scaler = StandardScaler().fit(data_processed)
X = scaler.transform(data_processed)

#%% Antes resampleo base de datos

#Aplicación método de componenetes principales (PCA) para llevar el espacio a dos
#dimensiones
pca = PCA(n_components=2)
X = pca.fit_transform(X)

#Dar formato pandas dataframe a datos transformados
X = pd.DataFrame(X, columns=["Variable 1", "Variable 2"])
y = pd.DataFrame(y)

#Reordenar dataframe para que se grafiquen últimos los datos correspondientes a
#la falla y se aprecie bien el overlap entre las clases
index_saludable=list(range(0, 20400)) + list(range(20952, 26881))
index_falla=list(range(20400, 20952))
new_index=index_saludable + index_falla
X = X.reindex(new_index)
y = y.reindex(new_index)
y = y.to_numpy()

#Invertir valores etiquetas para que los puntos saludables se grafiquen verde y
#las fallas rojas (debido a naturaleza del colormap escogido)
y_colors=np.copy(y)
CHANGE_COLORS1=True
if CHANGE_COLORS1==True:
    for i in range(len(y_colors)):
        if y_colors[i]==0:
            y_colors[i]=1
        else:
            y_colors[i]=0

#Plotear gráfico de dispersión de los datos antes de su resampleo
sns.set()
scatter = plt.scatter(
    x=X["Variable 1"],
    y=X["Variable 2"],
    c=y_colors,
    cmap="RdYlGn",
    alpha=0.8,
    s=1.5
)

classes = ['Falla', 'Saludable']
```

```
plt.legend(handles=scatter.legend_elements()[0], labels=classes)
plt.title('Visualización datos originales')
plt.xlabel('Variable 1')
plt.ylabel('Variable 2')
plt.savefig('C:\\Users\\raimu\\Desktop\\Memoria\\Data\\Imagenes\\
    ↪ Scatter_Caso_base.png', dpi=300, bbox_inches='tight')
plt.show()

#%% Resampleo base de datos

#Definir método de resampleo a implementar y gráficar
resampler = NeighbourhoodCleaningRule()

#Aplicar metodo de resampleo
X_resampled, y_resampled = resampler.fit_resample(X, y)

#Aplicación método de componenetes principales (PCA) para llevar el espacio a dos
#dimensiones
pca = PCA(n_components=2)
X_resampled = pca.fit_transform(X_resampled)
title='PCA'

#Invertir valores etiquetas para que los puntos saludables se grafiquen verde y
#las fallas rojas (debido a naturaleza del colormap escogido)
CHANGE_COLORS2=True
if CHANGE_COLORS2==True:
    for i in range(len(y_resampled)):
        if y_resampled[i]==0:
            y_resampled[i]=1
        else:
            y_resampled[i]=0

#Dar formato pandas dataframe a datos transformados
X_resampled = pd.DataFrame(X_resampled, columns=["Variable 1", "Variable 2"])

#Plotear gráfico de dispersión de los datos después de su resampleo
scatter = plt.scatter(
    x=X_resampled["Variable 1"],
    y=X_resampled["Variable 2"],
    c=y_resampled,
    cmap="RdYlGn",
    alpha=0.8,
    s=1.5
)

classes = ['Falla', 'Saludable']
plt.legend(handles=scatter.legend_elements()[0], labels=classes)
plt.title('Resultados resampleo Neighbourhood Cleaning Rule')
plt.xlabel('Variable 1')
plt.ylabel('Variable 2')
plt.savefig('C:\\Users\\raimu\\Desktop\\Memoria\\Data\\Imagenes\\
    ↪ Scatter_NeighbourhoodCleaningRule.png', dpi=300, bbox_inches='tight')
plt.show()
```

B.2. Definición de conjuntos de entrenamiento y evaluación

Código B.2: Función para definir combinación de conjunto de entrenamiento y testeo.

```
import numpy as np

def train_test_split(X, y, label_i, label_f, test_i, r):

    """

    Define una combinación de conjunto de entrenamiento y testeo al definir una
    ventana de tiempo dentro para el conjunto de testeo.

    :param np.array X:
        base de datos normalizada.
    :param np.array y:
        etiquetas de los datos.
    :param int label_i:
        número de registro de inucio de etiquetas con fallas.
    :param int label_f:
        número de registro final de etiquetas con fallas.
    :param int test_i:
        número de registro de incio de ventana de tiempo.
    :param float r:
        porcentaje de datos que son parte del conjunto de testeo.

    """

    test_f=test_i+round((label_f-label_i)*r)

    #Obtener datos de testeo que no pertenezcan a la ventana
    labels_test=[]
    while len(labels_test)<(round(len(X)*r)-(test_f-test_i)):
        if np.random.random()<(np.mean([label_i,label_f])/len(X)):
            b=np.random.randint(0,label_i)
            if not b in labels_test:
                labels_test.append(b)
        else:
            c=np.random.randint(label_f+1,len(X))
            if not c in labels_test:
                labels_test.append(c)

    #Unir etiquetas testeo
    labels_v=list(range(test_i,test_f))
    labels_test=list(np.concatenate((labels_test,labels_v)))
    labels_test.sort()

    #Definir etiquetas entrenamiento
    labels_train=list(range(len(X)))
    for i in range(len(labels_test)):
        labels_train.remove(labels_test[i])

    #Filtar base de datos según etiquetas obtenidos
    X_train = X[labels_train]
    y_train = y[labels_train]

```

```
    X_test = X[labels_test]
    y_test = y[labels_test]

    return X_train, y_train, X_test, y_test
```

B.3. Ajuste de hiperparámetros

Código B.3: Función para buscar hiperparámetros óptimos mediante RandomizedSearchCV.

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import f1_score

def RandomizedSearchCV_modified(X_train, y_train, X_test, y_test, param_space
    ↪ ,
                                resample, title):

    """

    Automatiza la implementación de la herramienta RandomizedSearchCV de sklearn
    ↪ ,
    imprimiendo resultados y ploteando la matriz de confusión del mejor modelo
    entrenado.

    :param np.array X_train:
        conjunto de datos de entrenamiento (solo variables físiscas).
    :param np.array y_train:
        conjunto de datos de entrenamiento (solo etiquetas).
    :param np.array X_test:
        conjunto de datos de evaluación (solo variables físiscas).
    :param np.array y_test:
        conjunto de datos de evaluación (solo etiquetas).
    :param dict param_space:
        diccionario con los posibles valores que puede tomar cada hiperparámetro
    :param model resample:
        método de resampleo a aplicar.
    :param str title:
        título del gráfico de la matriz de confusión

    """

    #Si es el caso base, no realizar resampleo
    if resample == None:
        X_train_resampled, y_train_resampled = X_train, y_train
    else:
        #Aplicar metodo de resampleo
        X_train_resampled, y_train_resampled = resample.fit_resample(X_train,
    ↪ y_train)

    #Definir clasificador a entrenar
    model = MLPClassifier(random_state=0, max_iter=200)

```

```
    #Realizar búsqueda aleatoria de hiperparámetros
    model = RandomizedSearchCV(model, param_space, cv=5, n_iter=25, scoring =
    ↪  'f1')
    model.fit(X_train_resampled, y_train_resampled.ravel())

    #Imprimir resultados de la búsqueda
    print('\nResultados', title)
    print('\nBest parameters found:', model.best_params_)
    print('Best scores found:', model.best_score_)

    #Calculo f1-score de evaluación y entrenamiento respectivamente
    Yp = model.predict(X_test)
    y_predicted_train = model.predict(X_train_resampled)

    print('\nf1 score training:', f1_score(y_train_resampled, y_predicted_train))
    print('f1 score test:', f1_score(y_test, Yp))

    # Plotear matriz de confusión (Falla=1 y Saludable=0)
    cm = confusion_matrix(y_test, Yp,labels=model.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=model.classes_)
    disp.plot()
    plt.title('Matriz de confusión ' + title)
    plt.show()
```

B.4. Entrenamiento y evaluación de modelos

Código B.4: Script para entrenar y evaluar modelos.

```
import pandas as pd
import numpy as np

from imblearn.over_sampling import *
from imblearn.under_sampling import *
from imblearn.combine import SMOTEENN, SMOTETomek
from imblearn.metrics import sensitivity_score, specificity_score,
    ↪ geometric_mean_score

from imblearn.ensemble import BalancedBaggingClassifier
from imblearn.ensemble import BalancedRandomForestClassifier
from imblearn.ensemble import RUSBoostClassifier
from imblearn.ensemble import EasyEnsembleClassifier

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score, precision_score
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier

# import xgboost as xgb
import joblib

import os
import sys
PROJECT_ROOT = os.path.abspath(os.path.join(
```

```
                os.path.dirname(__file__),
                os.pardir)
)
sys.path.append(PROJECT_ROOT)
import utils

#Cargar datos
data = pd.read_csv('C:/Users/raimu/Desktop/Memoria/data/Bba118.csv')
data = data.set_index('Unnamed: 0')

# IMPORTANTE: En este punto del código se lleva a cabo el preprocesamiento de los datos, pero debido a que incluye material proporcionado por una empresa, no se puede incluir.

#Definir posición inicio y fin etiquetas respectivamente
label_i=20400
label_f=20951

#Definir posición inicio ventana de tiempo a evaluar
test_i=20420

#Definir vector de etiquetas
y=np.zeros((len(data_processed),1))
y[label_i:label_f]=1

#Transformar a numpy array para poder aplica scaler
data_processed_pd = data_processed
data_processed = data_processed.to_numpy()

#Estandarización datos/Aplicación scaler
scaler = StandardScaler().fit(data_processed)
X = scaler.transform(data_processed)

#Guardar modelo scaler
joblib.dump(scaler,'C:\\Users\\raimu\\Desktop\\Memoria\\Data\\Modelos guardados\\Scaler_Bomba_118.pkl')

#Cantidad de combinaciones de conjuntos de entrenamiento y testeo a evaluar
n_iter=10

#Cantidad de modelos a evaluar (en este ejemplo solo son 2, pero pueden ser más)
n_models=2

#Matrices para guardar resultados
Resultados=np.zeros(shape=(5*n_models,n_iter))
Resultados_entrenamiento=np.zeros(shape=(n_models,n_iter))

#Iterar una cantidad de veces n_iter=10
for i in range(n_iter):

    #Definir los conjuntos de entrenamiento y testeo
    X_train, y_train, X_test, y_test = utils.train_test_split(X, y, label_i=label_i,
                                            label_f=label_f, test_i=test_i, r=0.25)

    #-----------------------------------------------------------------------
    #Si es que se desea aplicar un método de resampleo, incluir estas lineas de código,
    #si se quiere estudiar el caso base, se deben comentar o eliminar.

```

```
    resample = RandomOverSampler() #Cambiar según el método de resampleo de interés
    X_train, y_train = resample.fit_resample(X_train, y_train)
    #-----------------------------------------------------------------------

    #Definir red neuronal multicapa para el caso base
    model = MLPClassifier(random_state=0, hidden_layer_sizes=35, alpha=0.001)

    #Entrenar modelo
    model = model.fit(X_train, y_train.ravel())

    #Utilizar modelo entrenado para predecir etiquetas datos de entrenamiento y evaluación
    Yp=model.predict(X_test)
    y_predicted_train = model.predict(X_train)

    #Calcular y guardar métricas de los resultados obtenidos
    Resultados_entrenamiento[0,i] = f1_score(y_train, y_predicted_train)
    Resultados[0,i]=f1_score(y_test, Yp)
    Resultados[1,i]=sensitivity_score(y_test, Yp)
    Resultados[2,i]=specificity_score(y_test, Yp)
    Resultados[3,i]=geometric_mean_score(y_test.ravel(), Yp)
    Resultados[4,i]=precision_score(y_test, Yp)

    #Si el f1-score del modelo entrenado es el mejor de las iteraciones hasta el momento, entonces guardar modelo para poder utilizarlo después
    if Resultados[0,i] == max(Resultados[0]):
        joblib.dump(model, 'C:\\Users\\raimu\\Desktop\\Memoria\\Data\\Modelos guardados\\Mejor_Modelo_CB.pkl')

    #Se puede cargar un modelo guardado para utilizarlo de la misma manera que los otros
    model_load = joblib.load('C:\\Users\\raimu\\Desktop\\Memoria\\Data\\Modelos guardados\\Mejor_Modelo_CB.pkl')
```