UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MÁSTER
ESTADÍSTICA APLICADA

Evaluación de técnicas de preprocesamiento para problemas de clasificación con datos desequilibrados

Autor
David Núñez Nepomuceno

Directores
Pedro María Carmona Sáez
José Antonio Sáez Muñoz

Facultad de Ciencias
—
Granada, Junio de 2024

Evaluación de técnicas de preprocesamiento para problemas de clasificación con datos desequilibrados

David Núñez Nepomuceno

Palabras clave: clasificación, desequilibrio de clases, datos con ruido, filtros del ruido, datos transcriptómicos.

Resumen

La clasificación de datos consiste en asignar etiquetas o categorías a nuevas observaciones basándose en datos previamente etiquetados. Sin embargo, a pesar de los avances en las técnicas de clasificación, este problema presenta varios desafíos significativos. El desequilibrio de clases ocurre cuando algunas de ellas están subrepresentadas, lo que dificulta la clasificación correcta de estas clases minoritarias. Además, el ruido en los datos puede introducir errores y reducir la precisión de los modelos.

En este trabajo se estudia la eficacia de técnicas de procesamiento para mejorar la predicción de resultados en problemas de clasificación de datos desequilibrados y ruidosos. Se ha utilizado una base de datos de expresión genética de 200 genes en 996 individuos, de los cuales 924 son casos de lupus eritematoso sistémico, una enfermedad autoinmune, y 74 son controles sanos. La naturaleza desequilibrada y ruidosa de estos datos requiere técnicas de balanceo de clases y filtrado de ruido para mejorar los modelos de clasificación. Para abordar estos desafíos, se aplicó y comparó la técnica de balanceo de clases *Synthetic Minority Over-sampling Technique* (SMOTE) y dos métodos de filtrado de ruido: *Edited Nearest Neighbors* (ENN) e *Iterative Partitioning Filter* (IPF). Se seleccionaron tres algoritmos de clasificación: el algoritmo C4.5, *Random Forest* y máquina de vectores de soporte (SVM, de sus siglas en inglés). Para validar la robustez de los modelos, se empleó la técnica de validación cruzada estratificada de 5 particiones. Finalmente, las métricas de evaluación utilizadas fueron la exactitud, la precisión, la sensibilidad, la especificidad, la media geométrica, el *F1 score*, el área bajo la curva ROC y el coeficiente de correlación de Matthews.

En general, al aplicar técnicas de preprocesamiento, se observan diferentes efectos en los algoritmos de clasificación. Con C4.5, SMOTE mantiene la exactitud, pero disminuye con ENN e IPF, aunque el AUC y la sensibilidad mejoran con todas las técnicas, y el *F1 score* y el MCC tienen variaciones. En *Random Forest*, la exactitud y el *F1 score* mejoran con SMOTE, pero disminuyen con ENN, mientras que IPF las mantiene altas; el AUC se mantiene constante y la sensibilidad mejora con todas las técnicas, aunque la especificidad disminuye ligeramente. En SVM, tanto la exactitud como el *F1 score* y el MCC mejoran con SMOTE e IPF, disminuyendo con ENN, mientras que el AUC y la sensibilidad mejoran con todas las técnicas y la especificidad muestra variaciones.

Con los resultados obtenidos puede afirmarse que la combinación de técnicas de balanceo, como

SMOTE, con métodos de filtrado de ruido puede mejorar significativamente el rendimiento de los modelos de clasificación en los datos estudiados. Específicamente, en situaciones donde es crucial predecir correctamente la clase minoritaria, el uso de estas técnicas de preprocesamiento mejora notablemente los resultados.

Evaluation of preprocessing techniques for classification problems with imbalanced data

David Núñez Nepomuceno

Keywords: classification, class imbalance, noisy data, noise filters, transcriptomic data.

Abstract

Data classification involves assigning labels or categories to new observations based on previously labeled data. However, despite advances in classification techniques, several significant challenges remain. Class imbalance occurs when some classes are underrepresented, making it difficult to correctly classify these minority classes. Additionally, noise in the data can introduce errors and reduce model accuracy.

This work studies the effectiveness of preprocessing techniques to improve outcome prediction in noisy and imbalanced classification problems. We used a database with gene expression measurements of 200 genes in 996 individuals, of which 924 are cases of systemic lupus erythematosus, an autoimmune disease, and 74 are healthy controls. The imbalanced and noisy nature of these data requires class balancing and noise filtering techniques to improve the classification models. To address these challenges, the class balancing technique *Synthetic Minority Over-sampling Technique* (SMOTE) and two noise filtering methods, *Edited Nearest Neighbors* (ENN) and *Iterative Partitioning Filter* (IPF), were applied and compared. Three classification algorithms were selected: the C4.5 algorithm, *Random Forest*, and *Support Vector Machine* (SVM). To validate the robustness of the models, a 5-fold stratified cross-validation technique was used. Finally, the evaluation metrics used were accuracy, precision, sensitivity, specificity, geometric mean, *F1 score*, area under the ROC curve, and Matthews correlation coefficient.

In general, applying preprocessing techniques results in different impacts on the classification algorithms. With C4.5, SMOTE maintains accuracy but decreases with ENN and IPF, although AUC and sensitivity improve with all techniques, and *F1 score* and MCC vary. In *Random Forest*, accuracy and *F1 score* improve with SMOTE but decrease with ENN, while IPF maintains them at high levels; AUC remains constant, and sensitivity improves with all techniques, although specificity decreases slightly. In SVM, both accuracy and *F1 score* and MCC improve with SMOTE and IPF, decreasing with ENN, while AUC and sensitivity improve with all techniques, and specificity shows variations.

With the results obtained, we can affirm that the combination of noise filtering and balancing techniques like SMOTE can significantly improve the performance of classification models on the dataset studied. Specifically, in situations where correctly predicting the minority class is crucial, the use of these preprocessing techniques notably enhances the results.

Agradecimientos

Me gustaría agradecer a mis supervisores, Pedro María Carmona Sáez y José Antonio Sáez Muñoz, por su apoyo durante mi Trabajo de Fin de Máster. Gracias por vuestros consejos prácticos y vuestra atención al detalle que han dado forma a este trabajo.

Estoy muy agradecido por todo lo que he aprendido de ambos y espero llevar estos aprendizajes conmigo mientras avanzo en mi carrera. ¡Gracias por todo!

Índice

1. Introducción

La clasificación de datos es un problema central en el campo de la minería de datos (Krishnaiah et al., 2014), una disciplina académica interdisciplinar que combina principios y prácticas de las matemáticas, la estadística, la inteligencia artificial y la ingeniería de computación para analizar grandes cantidades de datos. Este proceso implica asignar categorías a nuevas observaciones basándose en un conjunto de datos previamente etiquetado. Las técnicas de clasificación se utilizan en una amplia variedad de aplicaciones, desde la detección de *spam* en correos electrónicos hasta el diagnóstico médico y la identificación de objetos en imágenes (Das et al., 2015).

A pesar de los avances en este campo, aún existen varios desafíos significativos (Fernández et al., 2017). La alta dimensionalidad de los datos (Donoho, 2000; Fernández et al., 2017; Rajender & Gopalachari, 2023; Ray et al., 2021) puede complicar el proceso de clasificación, ya que el aumento de las características puede llevar a una mayor complejidad computacional y al riesgo de sobreajuste. Además, el desequilibrio de clases (Fernández et al., 2017; Ghosh et al., 2022; Gosain & Sardana, 2017; Guzmán-Ponce et al., 2020; Johnson & Khoshgoftaar, 2019) es un problema común en muchos dominios, donde algunas clases están subrepresentadas, lo que dificulta la correcta clasificación de estas clases minoritarias. Por último, el ruido en los datos (Fernández et al., 2017; Hong et al., 2022; Moutsopoulos et al., 2021; Sáez et al., 2016; Xiong et al., 2006), que es frecuente en datos reales, puede introducir errores y reducir la precisión de los modelos, complicando aún más el proceso de clasificación confiable.

Los datos desequilibrados ocurren cuando las clases representadas en el conjunto de datos tienen una distribución desigual, es decir, una o más clases tienen significativamente más muestras que otras. Este desequilibrio puede llevar a que los modelos predictivos desarrollen un sesgo hacia las clases mayoritarias, ignorando las minoritarias y, por ende, disminuyendo la capacidad de predicción de estas últimas (Ghosh et al., 2022; Guzmán-Ponce et al., 2020; Pradipta et al., 2021). Para abordar este problema, una técnica comúnmente utilizada es *Synthetic Minority Over-sampling Technique* (SMOTE) (Chawla et al., 2002). SMOTE funciona generando nuevas instancias sintéticas de la clase minoritaria mediante la interpolación entre instancias existentes. Esto se realiza seleccionando un punto de datos de la clase minoritaria y luego creando puntos sintéticos a lo largo de la línea que conecta este punto con uno de sus vecinos más cercanos. Al aumentar el número de instancias en la clase minoritaria, SMOTE ayuda a equilibrar la distribución de clases, mejorando así la capacidad del modelo para predecir con mayor precisión las clases minoritarias.

Por su parte, los datos ruidosos se refieren a instancias de una clase que están ubicadas dentro del área correspondiente a otra clase. El impacto del ruido es profundo, ya que puede llevar a modelos

sobreajustados, predicciones incorrectas y, en última instancia, a decisiones basadas en información errónea (Gupta & Gupta, 2019; Liang et al., 2022; Nettleton et al., 2010). Para mitigar los efectos del ruido en los datos, se utilizan técnicas de filtrado, como pueden ser *Edited Nearest Neighbors* (ENN) (Wilson, 1972)e *Iterative Partitioning Filter* (IPF) (Chen et al., 2016).

Por otro lado, en la investigación biomédica es cada vez más habitual el uso de datos ómicos que pueden presentar estos problemas comentados anteriormente. Estos datos son heterogéneos y de alta dimensión y son particularmente susceptibles al ruido, ya que los valores de un atributo varían entre réplicas biológicas e incluso células (Xie et al., 2021). También puede ser evidente el problema del desequilibrio de clases, por la dificultad en el reclutamiento de instancias de un determinado tipo o porque los diseños experimentales en muchas ocasiones no son óptimos, entre otros factores, por el coste que pueden llegar a tener este tipo de tecnologías. Utilizando una base de datos transcriptómicos como caso de estudio, se evaluará la eficacia del método de balanceo SMOTE y de los filtros de ruido ENN e IPF. El análisis de técnicas de preprocesamiento en la clasificación de datos transcriptómicos es de gran importancia debido al interés de estos datos en medicina e investigación biomédica (Tang et al., 2023; Tasci et al., 2022). Los sistemas de clasificación juegan un papel importante en el apoyo a la toma de decisiones, ya que permiten automatizar y acelerar el proceso de análisis de datos. En la clasificación de estos datos, donde la calidad de los datos es crucial para la identificación correcta de fenotipos, patologías, o para la personalización de tratamientos médicos, el ruido puede distorsionar significativamente los resultados y llevar a interpretaciones erróneas (Hong et al., 2022; Vukadinovic et al., 2024). Además, estos conjuntos de datos presentan características, como la alta dimensionalidad y problemas de desequilibrio de clases, pues no se suelen tener datos balanceados de pacientes sanos y enfermos.

Este Trabajo de Fin de Máster tiene como objetivo principal analizar y evaluar la eficacia de diversas técnicas de balanceo de datos y filtrado de ruido aplicadas a la clasificación de datos transcriptómicos. Para ello, se centra en la aplicación y comparación de una técnica de balanceo de clases, como es SMOTE, y dos métodos específicos de filtrado de ruido: ENN e IPF. Además, se utilizarán tres algoritmos de clasificación ampliamente utilizados en el análisis de datos: el algoritmo C4.5 (Quinlan, 1992), el método de *Random Forest* (Breiman, 2001) y las máquinas de vectores de soporte (SVM) (Boser et al., 1992). La combinación de estas técnicas de preprocesamiento y clasificación será evaluada para determinar su impacto en la robustez de los modelos de clasificación de datos.

Para esta evaluación se realizará una revisión exhaustiva de diversas medidas de rendimiento. Entre las métricas consideradas se incluyen la exactitud, la precisión, la sensibilidad, la especificidad, la media geométrica, el *F1 Score*, el área bajo la curva (AUC) y el coeficiente de correlación de Matthews (MCC). Es importante destacar que la media geométrica, el *F1 Score*, el AUC y el MCC son

especialmente útiles en escenarios con datos no balanceados, ya que proporcionan una mejor comprensión del rendimiento del modelo en la predicción de la clase minoritaria.

El resto de este Trabajo de Fin de Máster se estructura como sigue. En la Sección 2, se describen los Objetivos del trabajo. La Sección 3 se dedica a los Algoritmos de clasificación, describiendo tres métodos clave: el Algoritmo de generación de árboles de decisión C4.5 (Subsección 3.1), el Método de *ensembles Random Forest* (Subsección 3.2) y las Máquinas de vectores soporte (Subsección 3.3). A continuación, la Sección 4 aborda la Clasificación no balanceada, presentando diversas técnicas para enfrentar este desafío, como los Métodos de *oversampling* (Subsección 4.1) y las Técnicas de filtrado del ruido (Subsección 4.2). La Sección 5 se enfoca en las Métricas de evaluación de modelos de clasificación. La Sección 6 describe el Marco experimental, incluyendo la Base de datos utilizada (Subsección 6.1) y la Metodología de análisis aplicada (Subsección 6.2). La Sección 7 presenta los Resultados, donde se discuten los hallazgos más significativos del estudio. Finalmente, la Sección 8 expone las Conclusiones con los principales aportes de la investigación y sugiriendo posibles líneas futuras de trabajo.

2. Objetivos

El presente Trabajo de Fin de Máster tiene como objetivo principal analizar y evaluar la eficacia de diversas técnicas de balanceo de datos y filtrado de ruido aplicadas a la clasificación de datos transcriptómicos. Con este objetivo central, se han definido varios objetivos específicos que guiarán el desarrollo del trabajo:

- Estudiar diversas técnicas de preprocesamiento de datos desequilibrados, incluyendo técnicas de remuestreo, *oversampling* y *undersampling*, para abordar el problema del desequilibrio de clases en el análisis de datos.
- Aplicar el método de balanceo de clases SMOTE en un conjunto de datos con desequilibrio de clases.
- Aplicar técnicas de filtrado de ruido, específicamente ENN e IPF, para reducir el impacto de instancias ruidosas en los modelos de clasificación.
- Estudiar y comparar tres algoritmos de clasificación ampliamente utilizados: C4.5 (algoritmo de generación de árboles de decisión), *Random Forest* (método de *ensemble*) y Máquinas de vectores de soporte.
- Realizar una revisión exhaustiva de diversas métricas de rendimiento para evaluar la eficacia de los modelos de clasificación. Las métricas consideradas incluyen exactitud, precisión, sensibilidad, especificidad, media geométrica, *F1 Score*, área bajo la curva y el coeficiente de correlación de Matthews.
- Extraer conclusiones sobre las medidas de evaluación más adecuadas para modelos de clasificación aplicados a datos transcriptómicos desequilibrados.
- Analizar el impacto de las técnicas de preprocesamiento (SMOTE, ENN e IPF) en el desempeño de los modelos de clasificación, particularmente en la capacidad de predecir la clase minoritaria.
- Utilizar una base de datos transcriptómicos como caso de estudio para evaluar la eficacia de las técnicas de balanceo y filtrado de ruido en un contexto real.

3. Algoritmos de clasificación

Las técnicas de clasificación constituyen una herramienta indispensable dentro del campo de la minería de datos, permitiendo la organización y categorización de datos en conjuntos o clases basadas en características específicas observadas (Kotsiantis, 2007). La clasificación supervisada (Jiang et al., 2020; Nasteski, 2017; Singh et al., 2016) es una de las tareas que más frecuentemente son llevadas a cabo por los denominados algoritmos de clasificación. Así pues, existen múltiples paradigmas para desarrollar algoritmos capaces de realizar tareas de clasificación (Jiang et al., 2020; Nasteski, 2017; Singh et al., 2016). Por ejemplo, C4.5 (Quinlan, 1992) es un algoritmo que sigue el paradigma de árboles de decisión. *Random Forest* (Breiman, 2001) se engloba dentro de los denominados métodos de *ensemble*, ya que combina múltiples arboles de decisión para construir el modelo. Por otro lado, SVM (Boser et al., 1992) pertenece un paradigma particular, en el que se busca encontrar el hiperplano que mejor separa las clases en el espacio. Este trabajo aborda la aplicación y evaluación de estas técnicas de clasificación con el fin de identificar la presencia o ausencia de lupus eritematoso sistémico en pacientes, enfrentándose a retos particulares como el desequilibrio de clases y el ruido en los datos.

3.1. Algoritmo de generación de árboles de decisión C4.5

Los árboles de decisión son modelos que no requieren la estimación de parámetros de una ecuación propuesta como en modelos estadísticos tradicionales (Lemon et al., 2003; Loh, 2011; Salzberg, 1994). Utilizan un método de particiones sucesivas para categorizar los datos. Este enfoque, también conocido como segmentación jerárquica (Elmaizi et al., 2022), empieza con una variable dependiente y segmenta los datos en grupos homogéneos utilizando combinaciones de variables independientes. Este proceso de división secuencial, iterativo y descendente, no solo es eficaz para tratar con grandes volúmenes de datos, sino que también revela patrones complejos que métodos convencionales, como la regresión, pueden no detectar (Lemon et al., 2003; Loh, 2011).

En la construcción de un árbol de decisión, se comienza por un nodo inicial, llamado nodo raíz, y se procede a dividir la muestra basándose en el valor de una variable independiente, elegida de tal manera que maximice la homogeneidad de los nuevos subconjuntos. Este proceso se repite en cada nodo nuevo, seleccionando variables y puntos de corte adecuados, hasta que todos los datos estén correctamente clasificados (Lemon et al., 2003; Loh, 2011). La Figura 1 presenta la estructura genérica de un árbol de decisión.

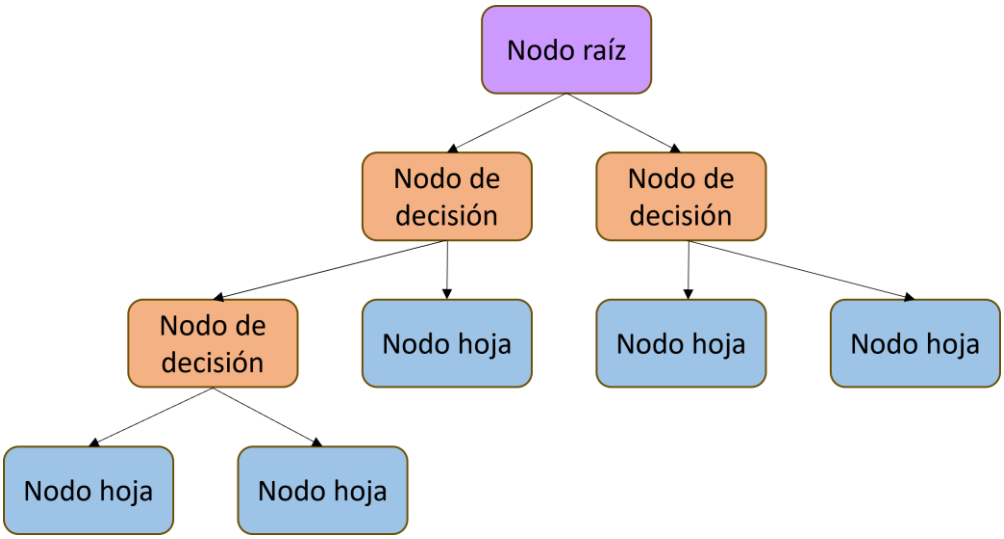


Figura 1. Diagrama de un árbol de decisión. El diagrama muestra la estructura de un árbol de decisión, comenzando con el nodo raíz en la parte superior, seguido por varios nodos de decisión y terminando en nodos hoja. Los nodos de decisión representan puntos donde se toma una decisión basada en ciertos criterios, y los nodos hoja representan los resultados finales de esas decisiones.

Los componentes principales de los árboles de decisión son los nodos, que representan variables de entrada; las ramas, que corresponden a los valores que estas variables pueden tomar; y las hojas, que representan los resultados de la clasificación. El nodo raíz, que inicia el árbol, es la variable de mayor relevancia en el proceso de clasificación. Sin embargo, un desafío común en el uso de árboles de decisión es el sobreajuste, especialmente si el modelo se ajusta demasiado a las incoherencias presentes en los datos de entrenamiento. Para mitigar este problema, se utilizan técnicas de poda (Lemon et al., 2003; Loh, 2011).

Las reglas de parada, como la pureza de nodo, la cota de profundidad y el umbral de soporte, son cruciales para controlar el crecimiento del árbol y asegurar que su complejidad no comprometa su generalidad y efectividad. La poda puede realizarse mediante el método de *coste-complejidad*, que busca un equilibrio entre la precisión y el tamaño del árbol, o mediante la *poda pesimista*, que elimina subárboles que no mejoran significativamente la precisión del clasificador (Lemon et al., 2003; Loh, 2011).

C4.5 (Quinlan, 1992) es un algoritmo avanzado de árbol de decisión, desarrollado por Ross Quinlan en 1992 como una extensión de su predecesor ID3 (Quinlan, 1986). Este algoritmo emplea la ganancia de información, derivada de la entropía, para seleccionar el mejor atributo en cada nodo del árbol. La entropía se calcula para el conjunto de datos *S* usando la fórmula:

$$\text{Entropía}(S) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

donde p_i es la proporción de la clase i en el conjunto S, y c es el número de clases.

La ganancia de información se calcula entonces como la reducción en la entropía después de dividir el conjunto S en subconjuntos usando un atributo A:

$$\text{Ganancia}(S, A) = \text{Entropía}(S) - \sum_{v \in \text{Valores}(A)} \frac{|S_v|}{|S|} \text{Entropía}(S_v)$$

donde S_v representa el subconjunto de S para el valor v del atributo A.

C4.5 construye el árbol de decisión de manera recursiva dividiendo el conjunto de datos, seleccionando en cada paso el atributo que ofrece la mayor ganancia de información. La construcción continúa hasta que los puntos de datos en un nodo pertenecen a una sola clase o hasta que no se pueda obtener más información de los atributos restantes. Posteriormente, el algoritmo realiza una poda del árbol para eliminar las ramas que no contribuyen significativamente a la mejora del modelo (Quinlan, 1992; Salzberg, 1994).

La poda pesimista (Mohamed et al., 2012) es una innovación en C4.5. Consiste en estimar el error que podría ocurrir basado en el número de clasificaciones erróneas en el conjunto de entrenamiento. Este enfoque estima recursivamente la tasa de error asociada con un nodo basada en las tasas de error estimadas de sus ramas. Para una hoja con N instancias y E errores (es decir, el número de instancias que no pertenecen a la clase predicha por esa hoja), la poda pesimista primero determina la tasa de error empírica en la hoja como la relación $(E + 0.5)/N$. Para un subárbol con L hojas, E errores correspondientes y N instancias en estas hojas, se estima que la tasa de error para todo el subárbol es $(E + 0.5 \cdot L)/N$. Si este el subárbol es reemplazado por su mejor hoja y J es el número de casos del conjunto de entrenamiento que clasifica erróneamente, la poda pesimista reemplaza el subárbol con esta mejor hoja si $(J + 0.5)$ está dentro de una desviación estándar de $(E + 0.5 \cdot L)$. Este enfoque puede extenderse para podar basado en intervalos de confianza deseados. Podemos modelar las tasas de error *e* en las hojas como variables aleatorias de Bernoulli y, para un umbral de confianza CI dado, se puede determinar un límite superior e_{max} tal que $e < e_{max}$ con una probabilidad de $1-CI$ (C4.5 utiliza un IC predeterminado de 0.25). Puede irse aún más allá y aproximar e mediante la distribución normal (para N grandes), en cuyo caso C4.5 determina un límite superior del error esperado como:

$$e_{max} = \frac{e + \frac{z^2}{2N} + z\sqrt{\frac{e}{N} - \frac{e^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}$$

donde e es la tasa de error, z se elige basado en el intervalo de confianza deseado para la estimación, asumiendo una variable aleatoria normal con media cero y varianza unitaria, es decir, $N(0,1)$.

C4.5 puede procesar tanto atributos continuos como categóricos. Para los atributos continuos, como es el caso que se trata en este trabajo, el algoritmo busca un umbral que divida los datos en grupos con la mayor pureza posible. Para atributos categóricos, cada categoría puede considerarse una división potencial. El algoritmo maneja valores faltantes al asignar probabilidades a cada posible valor de un atributo basado en su distribución en los datos no faltantes, permitiendo que la construcción del árbol continúe sin imputación directa.

3.2. Método de *ensembles Random Forest*

Random Forest (Breiman, 2001) es un algoritmo de clasificación que combina múltiples árboles de decisión para formar un modelo de *ensemble* más robusto y preciso. Este método, desarrollado por Leo Breiman y Adele Cutler en 2001, es especialmente eficaz en la mitigación de problemas de sobreajuste comunes en modelos más simples. Cada árbol en el bosque se entrena usando una muestra diferente del conjunto de datos original, seleccionada con reemplazo. Este proceso se denomina muestreo de *bootstrap*. La fórmula para el muestreo es:

$$S_i \sim \text{Bootstrap}(D),$$

donde S_i es el subconjunto de datos usado para entrenar el *i*-ésimo árbol, y D es el conjunto de datos original. En cada nodo del árbol, se selecciona un número limitado de características al azar de las disponibles. Este número es m. Esta estrategia asegura que los árboles sean descorrelacionados y aumenta la diversidad en el modelo:

$$A_i \subset F,\ |A_i| = m,$$

donde A_i es el conjunto de características seleccionadas para el *i*-ésimo árbol, y F es el conjunto total de características. La decisión final del modelo *Random Forest* se toma agregando las predicciones de todos los árboles individuales. Este proceso se denomina *Bagging.* Para la clasificación, se usa la moda de las predicciones:

$$\hat{y} = \text{moda}\big(f_1(x), f_2(x), \dots, f_N(x)\big).$$

La técnica de muestreo de *bootstrap* junto con la selección aleatoria de características en cada nodo ayuda a reducir el riesgo de sobreajuste, común en árboles de decisión individuales. Este algoritmo maneja datos faltantes asignando valores basados en la proximidad de otros puntos de datos o la imputación dentro de cada árbol. La varianza de las predicciones se reduce significativamente debido al promedio de múltiples árboles. El algoritmo proporciona una evaluación directa de la importancia de cada característica basada en cuánto contribuye a la mejora de la pureza de los nodos, medido generalmente por la ganancia de información o el índice de Gini. La ganancia de información se calcula, de manera similar al algoritmo C4.5, como la reducción en la entropía después de dividir el conjunto S en subconjuntos usando un atributo A:

$$\text{Ganancia}(S, A) = \text{Entropía}(S) - \sum_{v \in \text{Valores}(A)} \frac{|S_v|}{|S|} \text{Entropía}(S_v),$$

donde S_v representa el subconjunto de S para el valor v del atributo A.

Por su parte, el índice Gini (Gastwirth, 1972) es una medida de impureza que determina cuán puro o impuro es un nodo después de una división. Un nodo es puro si todos sus elementos pertenecen a una sola clase. La fórmula para calcular el Índice Gini es:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

donde p_i es la proporción de elementos de clase i en el nodo y C es el número de clases. Un índice Gini de 0 indica impureza mínima con todos los elementos pertenecientes a una sola clase, mientras que un Índice Gini de 0.5 representa la máxima impureza en un escenario binario, con clases distribuidas equitativamente en el nodo.

3.3. Máquinas de vectores soporte

Las máquinas de vectores soporte (Boser et al., 1992) fueron desarrolladas por Vapnik en la década de 1990 y operan bajo el principio de transformar los datos de entrada, representados como vectores *p*-dimensionales, en un espacio donde se busca maximizar la separación entre dos categorías posibles.

El núcleo del enfoque SVM es la construcción de un hiperplano que actúa como la frontera de decisión entre las clases. Esta frontera se caracteriza por maximizar la distancia entre el hiperplano y los puntos más cercanos de cada clase. Este enfoque de margen máximo es lo que otorga a las SVM su eficacia, permitiendo una clasificación precisa incluso en espacios de alta dimensionalidad (Bhavsar & Panchal, 2012; Salcedo-Sanz et al., 2014).

Además, es importante destacar que el problema de SVM es un caso de optimización convexa, más específicamente, un problema de programación cuadrática, una subcategoría dentro de la programación no lineal (Gotoh & Uryasev, 2017). La optimización convexa asegura que cualquier mínimo local es también un mínimo global, lo que es crucial para la eficacia de los algoritmos SVM (Simonyan et al., 2014).

En términos prácticos, un SVM se enfrenta a desafíos significativos cuando los datos presentan más de dos categorías o cuando las fronteras entre clases no son lineales. Para superar estas dificultades, las SVM utilizan funciones *kernel*, que proyectan los datos a un espacio de mayor dimensión donde la separación lineal es factible. Esta técnica no solo mejora la capacidad de manejo de la no linealidad, sino que también permite abordar conjuntos de datos complejos que no pueden ser separados linealmente (Bhavsar & Panchal, 2012; Salcedo-Sanz et al., 2014).

Para una tarea de clasificación de dos clases linealmente separables, el objetivo del SVM es encontrar un hiperplano que separe dos clases de muestras con un margen máximo. En un conjunto de datos de entrenamiento compuesto por n puntos de la forma $(x_1, y_1), \ldots, (x_n, y_n)$, donde los y_i son 1 o -1 indicando la clase a la que pertenece el punto x_i, y cada x_i es un vector real de p dimensiones, se busca encontrar el hiperplano de margen máximo que divida el grupo de puntos x_i para los cuales $y_i = 1$ del grupo de puntos para los cuales $y_i = -1$. Este hiperplano se define de manera que la distancia entre el hiperplano y el punto más cercano x_i de cualquiera de los dos grupos sea máxima.

Cualquier hiperplano puede expresarse como el conjunto de puntos x que satisfacen la ecuación:

$$w^T x - b = 0,$$

donde w es el vector normal al hiperplano. Esto es similar a la forma normal de Hesse (Hartmann, 1999), salvo que w no es necesariamente un vector unitario. El parámetro $\frac{b}{||w||}$ determina el desplazamiento del hiperplano desde el origen a lo largo del vector normal w.

Si los datos de entrenamiento son linealmente separables, es posible seleccionar dos hiperplanos paralelos que separen las dos clases de datos, de modo que la distancia entre ellos sea máxima (Boser et al., 1992). La región delimitada por estos dos hiperplanos se llama el margen, y el hiperplano de margen máximo es el hiperplano que se encuentra a la mitad de ellos. Con un conjunto de datos normalizado estos hiperplanos pueden describirse mediante las ecuaciones:

$w^T x - b = 1$ (todo lo que esté en o por encima de este límite es de una clase, con etiqueta 1),
$w^T x - b = -1$ (lo que esté en o por debajo de este límite es de la otra clase, con etiqueta -1).

Geométricamente, la distancia entre estos dos hiperplanos es $\frac{2}{||w||}$, por lo que, para maximizar la distancia entre los planos, se desea minimizar $||w||$. La distancia se calcula utilizando la ecuación de la distancia de un punto a un plano. Además, para evitar que los puntos de datos caigan en el margen, se añade una de las siguientes restricciones (para cada i):

$$w^T x_i - b \geq 1, \text{si } y_i = 1,$$
$$w^T x_i - b \leq -1, \text{si } y_i = -1.$$

Esto puede reescribirse como:

$$y_i(w^T x_i - b) \geq 1, \text{para todo } 1 \leq i \leq n.$$

Se puede formular este problema como un problema de optimización de la siguiente manera:

$$Min_{\mathbf{w},b}\left|\left|w\right|\right|_2^2 \text{ sujeto a } \quad y_i\,(w^T\,x_i - b) \geq 1 \quad \forall\, i \in \{1, \ldots, n\}.$$

Los w y b que resuelven este problema determinan el clasificador final, $x \mapsto \text{sgn}(w^T\,x - b)$, donde sgn(·)es la función signo. La Figura 2 ilustra el procedimiento para la clasificación de datos utilizando un clasificador SVM. Una consecuencia importante de esta descripción geométrica es que el hiperplano de margen máximo está completamente determinado por aquellos x_i que están más cerca de él. Estos x_i se llaman *vectores de soporte*.

En 1992, Bernhard Boser, Isabelle Guyon y Vladimir Vapnik sugirieron una manera de crear clasificadores no lineales aplicando el *truco del Kernel* a los hiperplanos de margen máximo. El truco del *Kernel* consiste en reemplazar los productos escalares por un *Kernel* adecuado. Esto permite que el algoritmo ajuste el hiperplano de margen máximo en un espacio transformado. La transformación puede ser no lineal y el espacio transformado de alta dimensión; aunque el clasificador es un hiperplano en el espacio transformado, puede ser no lineal en el espacio de entrada original. Algunos *Kernels* comunes incluyen:

- Polinomial (homogéneo): $k(x_i, x_j) = (x_i \cdot x_j)^d$. Particularmente, cuando $d = 1$, este se convierte en el *Kernel* lineal.
- Polinomial (no homogéneo): $k(x_i, x_j) = (x_i \cdot x_j + r)^d$.
- Función de base radial gaussiana: $k(x_i, x_j) = \exp(-\gamma|x_i - x_j|^2)$ para $\gamma > 0$. A veces parametrizado usando $\gamma = \frac{1}{2\sigma^2}$.

- Función sigmoidal (tangente hiperbólica): $k(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + c)$ para algunos $\kappa > 0$ y $c < 0$.

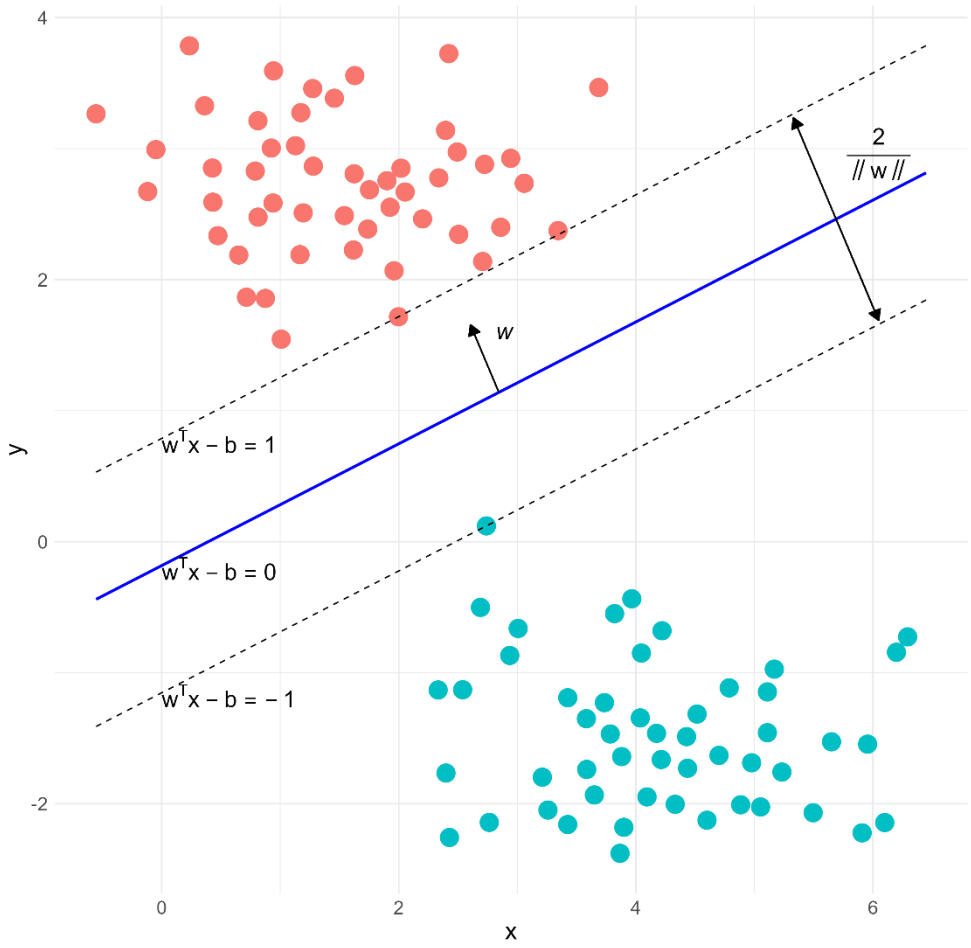


Figura 2. Diagrama de un clasificador SVM. La Figura muestra dos conjuntos de datos separados por un clasificador SVM. Los puntos en rojo representan una clase de datos, mientras que los puntos en azul representan otra clase. La línea de decisión, representada en azul marino, muestra el hiperplano que el SVM ha encontrado para separar las dos clases de datos en el espacio bidimensional, maximizando el margen entre los puntos de cada clase más cercanos al hiperplano (vectores de soporte).

4. Clasificación no balanceada

El desequilibrio de clases en el contexto del análisis de datos se refiere a la situación en la que el número de instancias en cada clase es muy diferente (Ghosh et al., 2022; Gosain & Sardana, 2017; Johnson & Khoshgoftaar, 2019). En otras palabras, una clase contiene la mayoría de las instancias mientras que la otra, que puede ser la clase más relevante, está representada por una proporción menor (Morales et al., 2020). Este fenómeno es común en los conjuntos de datos médicos, como los datos transcriptómicos (Tasci et al., 2022).

Los datos desequilibrados pueden resultar problemáticos porque la mayoría de los algoritmos de clasificación se diseñaron con la suposición de un número similar de instancias para cada clase (Ghosh et al., 2022). Como resultado, las clasificaciones realizadas a partir de un conjunto de datos desequilibrados suelen presentar más errores en las clases minoritarias.

Existen varias estrategias posibles para abordar el problema del desequilibrio de clases al clasificar datos genómicos (Hong et al., 2022; Morales et al., 2020). El *oversampling* consiste en aumentar la cantidad de instancias en la clase menos representada para igualarla con la clase más frecuente (Gosain & Sardana, 2017; Mohammed et al., 2020). El método más simple de *oversampling* es la duplicación aleatoria de muestras existentes en la clase minoritaria. Un método más sofisticado SMOTE (Chawla et al., 2002). SMOTE crea ejemplos sintéticos de la clase minoritaria al interpolar entre ejemplos existentes. Esto no solo aumenta el número de ejemplos en la clase minoritaria, sino que también introduce variabilidad. Existen además varias ampliaciones de SMOTE. Entre ellas se encuentran SMOTENC (Mukherjee & Khushi, 2021), que amplía SMOTE para conjuntos de datos mixtos; SMOTEN (Fonseca & Bacao, 2023), que es una variante de SMOTE que considera los *k* vecinos más cercanos; ADASYN (He et al., 2008), que se enfoca en regiones de alta densidad de la clase minoritaria; BorderlineSMOTE (Han et al., 2005), que se centra en muestras cercanas a la frontera de decisión; KMeansSMOTE (Fonseca et al., 2021), que utiliza centroides KMeans para generar muestras sintéticas; y SVMSMOTE (Tang et al., 2009), que emplea vectores de soporte para crear instancias sintéticas.

Por otro lado, el *undersampling* reduce el número de instancias en la clase más frecuente (Hulse et al., 2009). Esto se puede realizar eliminando muestras aleatoriamente para reducir el tamaño de la clase dominante hasta que las proporciones de clase sean más equitativas. Este método conlleva el riesgo de perder información importante, ya que se eliminan datos potencialmente útiles para la creación del modelo (Hulse et al., 2009). Existen técnicas más avanzadas de *undersampling* que intentan preservar la información más útil: *ClusterCentroids* (Singh et al., 2013), que reduce el tamaño de la muestra mediante la selección de centroides de *clusters*; *CondensedNearestNeighbour* (Filippakis

et al., 2023), que conserva la estructura general de la muestra eliminando instancias redundantes; ENN (Wilson, 1972), las cual elimina instancias que podrían introducir ruido o no aportar información relevante o IPF (Chen et al., 2016), que trata de ajustar una tabla de frecuencias observadas para que cumpla con ciertas restricciones marginales o totales, sin alterar la estructura general de los datos.

Para llevar a cabo este Trabajo Fin de Máster, se ha aplicado el método de *oversampling* SMOTE para equilibrar el número de observaciones de la clase *Healthy* en nuestros datos. Asimismo, se han utilizado los métodos de *undersampling* ENN e IPF para filtrar las observaciones que podrían introducir ruido en los datos. A continuación se recoge una explicación más en detalle sobre el funcionamiento de los métodos SMOTE, ENN e IPF.

4.1. Métodos de *oversampling*

El *oversampling* es una técnica utilizada en el procesamiento de conjuntos de datos desequilibrados (Gosain & Sardana, 2017; Mohammed et al., 2020). Esta técnica se orienta a equilibrar la distribución de clases mediante la adición de ejemplos a la clase subrepresentada. Entre las estrategias de *oversampling*, el método SMOTE es ampliamente utilizado en la investigación y la ciencia aplicada.

Propuesto por Chawla et al. (2002), SMOTE es un método que selecciona aleatoriamente una observación *x* de la clase minoritaria y encuentra sus k vecinos más cercanos en el mismo conjunto. Luego, selecciona uno de estos vecinos, x_{vecino}, y genera un nuevo ejemplo sintético *s* a lo largo de la línea que conecta *x* y x_{vecino}. La generación se realiza mediante la interpolación siguiente lineal:

$$s = x + \lambda \times (x_{vecino} - x)$$

donde λ es un número aleatorio entre 0 y 1, que determina el grado de interpolación entre los dos puntos. La principal ventaja de SMOTE radica en su capacidad para introducir diversidad en el conjunto de datos, creando observaciones sintéticas que ayudan a mejorar la generalización del modelo sin el riesgo de sobreajuste asociado a la duplicación simple de observaciones. La Figura 3 muestra una representación gráfica de la aplicación de SMOTE en un conjunto de datos. El grupo de puntos localizados a la izquierda en la Figura representan los datos de partida. Los puntos localizados en el centro muestran el proceso de generación de puntos de la clase minoritaria utilizando los datos existentes. Los datos localizados a la derecha representan el resultado final tras aplicar SMOTE.

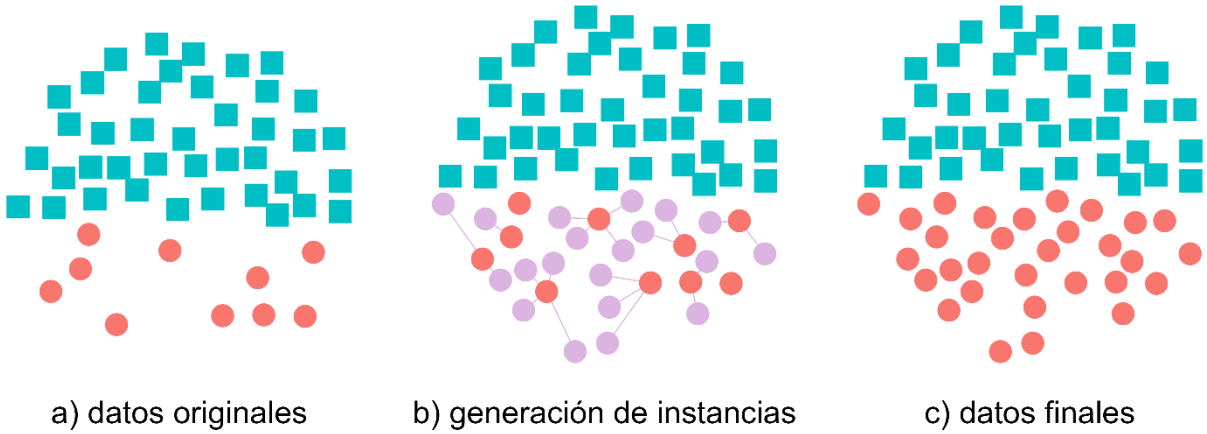


Figura 3: Representación gráfica del procedimiento SMOTE para el equilibrio de clases en conjuntos de datos. La Figura 3a muestra los datos originales con clases desequilibradas, donde los cuadrados azules son la clase mayoritaria y, los círculos naranjas, la minoritaria. La Figura 3b muestra el proceso de generación de puntos sintéticos de la clase minoritaria (círculos violetas) a partir de los existentes (círculos naranjas). La Figura 3c muestra el resultado final con una distribución más equilibrada entre las clases.

No obstante, el uso de SMOTE no está exento de desafíos (Fernández et al., 2018; Pradipta et al., 2021). La técnica puede generar ruido, especialmente si los ejemplos de la clase minoritaria están dispersos, ya que la interpolación puede producir puntos sintéticos que no reflejan características realistas de la clase. Además, la selección de vecinos inadecuados puede llevar a la generación de ejemplos que distorsionen la frontera de decisión del modelo, particularmente en casos de solapamiento significativo entre clases (Pradipta et al., 2021). Para mitigar estos posibles problemas, una posible estrategia consiste en combinar SMOTE con técnicas de filtrado de ruido como ENN e IPF.

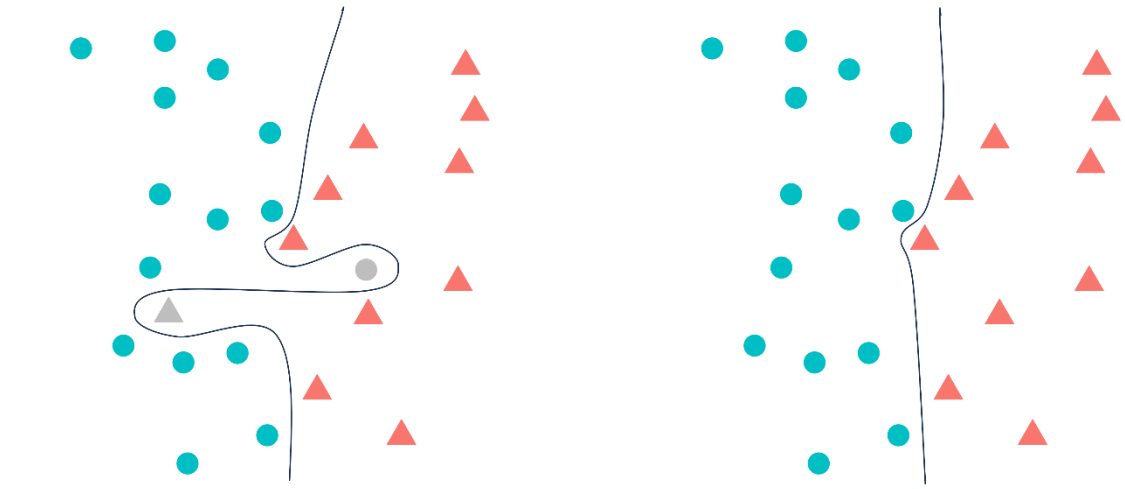
4.2. Técnicas de filtrado del ruido

Los datos ruidosos se refieren a instancias de una clase que se encuentran ubicados profundamente dentro de la región de la otra clase. Esto puede incluir instancias que contienen errores ya sea en los valores de los atributos o en la etiqueta de clase (Bello & Renter, 2018; Gupta & Gupta, 2019; Xiong et al., 2006). En 2015, Sáez et al., abordaron el problema del ruido en la clasificación de datos desequilibrados y desarrollaron el método SMOTE–IPF como un enfoque efectivo para enfrentar este desafío.

La Figura 4 ilustra el proceso general de filtrado de datos ruidosos. En ella se muestran dos escenarios de un conjunto de datos con dos clases. En la Figura 4a, el límite de decisión es complejo y trata de incluir todas las observaciones, incluidos los puntos grises que representan datos ruidosos. En la Figura 4b, después de eliminar los datos ruidosos, el límite de decisión es más simple y claro, separando eficazmente los círculos de los triángulos restantes.

En este estudio se emplean dos métodos de filtrado de los datos ruidosos. Estos son ENN e IPF, ambos dirigidos a mejorar la generalización del clasificador mediante la eliminación selectiva de

instancias ruidosas.



a) datos con puntos ruidosos sin filtrar b) datos filtrados

Figura 4. Conjunto de datos con dos instancias ruidosas. La Figura 4a muestra el límite de decisión creado al considerar las instancias con ruido como adecuadas. La Figura 4b presenta el límite de decisión después de la eliminación de los puntos ruidosos durante el proceso de filtrado. Adaptación de la Figura 1 en Sáez et al., 2016.

El método ENN fue introducido por Wilson en 1972. El proceso de ENN se puede describir en tres pasos principales: se comienza examinando cada punto dentro del conjunto de datos y analizando un grupo definido de sus vecinos más cercanos. Luego, se evalúa la consistencia de cada instancia, determinando si la mayoría de sus vecinos pertenecen a la misma clase que la instancia en cuestión. Finalmente, se procede a la eliminación de aquellas instancias cuyos vecinos mayoritariamente pertenecen a otras clases, bajo la suposición de que estas instancias son puntos de ruido o están mal clasificados.

Por su parte, IPF (Chen et al., 2016) es una técnica de filtrado de ruido que opera de manera iterativa para refinar el conjunto de datos eliminando instancias redundantes o ruidosas. El procedimiento de IPF se puede resumir en los siguientes pasos: se comienza con el particionamiento del conjunto de datos, donde se divide en varias particiones basadas en criterios de similaridad o proximidad entre los datos. A continuación, se lleva a cabo una evaluación iterativa de cada partición para determinar cómo los datos contenidos contribuyen a la precisión general del modelo. Durante cada iteración, se realiza una eliminación selectiva de las instancias que se identifican como los menos útiles o más problemáticas para una clasificación precisa. Al combinar SMOTE con ENN o IPF, los puntos sintéticos que no son consistentes con sus vecinos pueden eliminarse. Este proceso refina el conjunto de datos al eliminar datos potencialmente ruidosos y fortalecer la frontera de decisión, mejorando así la generalización del modelo.

5. Métricas de evaluación de modelos de clasificación

La selección de métricas de rendimiento adecuadas es crucial para evaluar la efectividad de un modelo. Sin embargo, algunas métricas pueden no proporcionar una visión completa del rendimiento del modelo cuando las clases están desequilibradas. A continuación, se describen varias métricas utilizadas para evaluar los algoritmos de clasificación y se comentan brevemente las limitaciones que presentan. En el artículo de revisión bibliográfica publicado por Hossin et al en 2015, se pueden consultar los detalles de estas métricas.

La matriz de confusión es una herramienta que muestra las combinaciones de clases reales y predichas. En este estudio consideramos como casos positivos a las instancias de la clase minoritaria y casos negativos a los de la clase mayoritaria. A continuación se presenta la estructura general de una matriz de confusión:

	Clase predicha positiva	**Clase predicha negativa**
Clase real positiva	TP	FN
Clase real negativa	FP	TN

Tabla 1: Matriz de confusión que muestra las cuatro combinaciones posibles de clases reales y predichas en un problema de clasificación binaria. Los verdaderos positivos (TP) y los verdaderos negativos (TN) representan las predicciones correctas, mientras que los falsos positivos (FP) y los falsos negativos (FN) representan las predicciones incorrectas.

Debemos mencionar que, en el contexto de métricas de rendimiento para la evaluación de modelos de clasificación, TP representa los verdaderos positivos, que son las instancias correctamente identificadas como positivas por el modelo. FP representa los falsos positivos, que son las instancias incorrectamente identificadas como positivas. FN representa los falsos negativos, que son las instancias que el modelo no identificó como positivas pero que en realidad lo son. Finalmente, TN representa los verdaderos negativos, que son las instancias correctamente identificadas como negativas por el modelo.

La *exactitud* mide la proporción de todas las predicciones que fueron correctas, tanto positivas como negativas. Se calcula como el número de verdaderos positivos más verdaderos negativos dividido por el total de predicciones realizadas. En conjuntos de datos desequilibrados, una alta exactitud puede ser engañosa, ya que el modelo puede simplemente predecir siempre la clase mayoritaria y aun así obtener una alta exactitud, sin capturar adecuadamente la clase minoritaria.

$$\text{Exactitud} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

La *precisión* indica la proporción de las predicciones positivas que son realmente positivas. Se calcula como el número de verdaderos positivos dividido por la suma de verdaderos y falsos positivos. En conjuntos de datos desequilibrados, un modelo puede tener una precisión alta simplemente prediciendo muy pocos positivos, incluso si hay muchas más instancias positivas en el conjunto de datos.

$$\text{Precisión} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

La *sensibilidad* y la *especificidad* son dos métricas cruciales en la evaluación de modelos de clasificación binaria. Sin embargo, es importante destacar que no deben ser estudiadas de manera aislada, ya que cada una solo considera los aciertos en una de las clases.

La *sensibilidad*, también llamada *recall* o tasa de verdaderos positivos (TVP), mide la capacidad del modelo para identificar correctamente los verdaderos positivos de entre todos los casos positivos reales. Es especialmente importante en contextos donde no detectar los positivos puede tener consecuencias graves (como enfermedades en diagnósticos médicos). Mientras que la sensibilidad es crítica para asegurar que todos los casos positivos sean detectados, no proporciona información sobre cómo el modelo maneja los negativos. Esto puede resultar en un modelo que identifica correctamente todos los casos positivos, pero a costa de clasificar incorrectamente muchos negativos como positivos.

$$\text{Sensibilidad} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

La *especificidad* mide la capacidad del modelo para identificar correctamente los verdaderos negativos. La especificidad solo considera los casos negativos. Un modelo puede tener una alta especificidad al evitar falsos positivos, pero esto puede llevar a un número elevado de falsos negativos, especialmente si el modelo es demasiado conservador.

$$\text{Especificidad} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

La *media geométrica* entre sensibilidad y especificidad se calcula como la raíz cuadrada del producto de ambas métricas. Esta métrica se utiliza para tener una medida más balanceada del rendimiento de

un clasificador, especialmente en casos de datos desequilibrados.

$$\text{Media geométrica} = \sqrt{\text{Sensibilidad} \times \text{Especificidad}}$$

El *F1 Score* es una medida que combina la precisión y la sensibilidad en un solo valor, utilizando su media armónica. Proporciona un balance entre la precisión y la sensibilidad, siendo útil en situaciones donde ambas son importantes.

$$F1\ Score = 2 \times \frac{\text{Precisión} \times \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}}$$

El *Área Bajo la Curva* (AUC) de la curva ROC (*Receiver Operating Characteristic*) es una de las métricas más utilizadas para evaluar el rendimiento de modelos de clasificación binaria. AUC mide la capacidad del modelo para discriminar entre clases bajo diferentes umbrales. Un AUC de 1 indica un modelo perfecto, mientras que un AUC de 0.5 sugiere un rendimiento no mejor que el azar. AUC proporciona una evaluación global del rendimiento del modelo.

$$\text{AUC} = \int_0^1 \text{TVP}(\text{FPR})\ d\text{FPR}$$

Calculándose la tasa de verdaderos positivos (TVP) y la tasa de falsos positivos (FPR) mediante las siguientes fórmulas:

$$\text{TVP} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

El *Coeficiente de Correlación de Matthews* (MCC) es una medida que toma en cuenta verdaderos y falsos y positivos y negativos, siendo considerado como un balance entre la sensibilidad y la especificidad. Es particularmente útil en contextos de clases desequilibradas porque proporciona una visión holística del rendimiento del modelo. Es una de las métricas más completas ya que considera todos los aspectos del rendimiento del modelo.

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}$$

A continuación, la Tabla 2 resume las métricas comentadas en esta Sección.

Métrica	**Fórmula**
Exactitud	$\text{Exactitud} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$
Precisión	$\text{Precisión} = \frac{\text{TP}}{\text{TP} + \text{FP}}$
Sensibilidad (*Recall*)	$\text{Sensibilidad} = \frac{\text{TP}}{\text{TP} + \text{FN}}$
Especificidad	$\text{Especificidad} = \frac{\text{TN}}{\text{TN} + \text{FP}}$
Media geométrica	$\text{Media geométrica} = \sqrt{\text{Sensibilidad} \times \text{Especificidad}}$
F1 Score	$\textit{F1 Score} = 2 \times \frac{\text{Precisión} \times \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}}$
Área bajo la curva	$\text{AUC} = \int_0^1 \text{TVP}(\text{FPR})\ d\text{FPR}$
Tasa de verdaderos positivos	$\text{TVP} = \frac{\text{TP}}{\text{TP} + \text{FN}}$
Tasa de falsos positivos	$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}$
Coeficiente de correlación de Matthews	$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}$

Tabla 2. Resumen de las métricas utilizadas para evaluar el desempeño de los modelos. La Tabla incluye las fórmulas para calcular la exactitud, la precisión, la sensibilidad, la especificidad, la media geométrica, el *F1 score*, el coeficiente de correlación de Matthews, el área bajo la curva, la tasa de verdaderos positivos y la tasa de falsos positivos.

6. Marco experimental

6.1. Base de datos

La base de datos analizada en este estudio contiene un total de 996 instancias, cada una correspondiente a una muestra individual. Estas muestras están descritas mediante 200 características numéricas, que representan mediciones de expresión genética. Además, cada entrada incluye una clasificación en una de dos posibles categorías: *Healthy* (sano) o *SLE* (lupus eritematoso sistémico). La distribución de las clases es desigual, con 924 muestras clasificadas como *SLE* y 72 como *Healthy*. Este desequilibrio presenta desafíos en el análisis estadístico y modelado predictivo, indicando la necesidad de estrategias de balanceo como SMOTE para manejar la desproporción en la clasificación.

Hemos realizado una serie de análisis exploratorios para obtener una mejor comprensión de la estructura y características inherentes de los datos. Estas técnicas permiten explorar la varianza explicada por las variables principales, identificar posibles agrupaciones y visualizar patrones.

Un análisis de componentes principales fue realizado para reducir la dimensionalidad de los datos y para identificar las dimensiones más significativas en términos de varianza explicada. La Figura 4 muestra el porcentaje de varianza explicada por cada componente principal. La línea roja representa la varianza acumulada explicada por las componentes. Las primeras dos componentes explican aproximadamente el 37.2% y el 17.7% de la varianza respectivamente (54.9% entre ambos), y con diez componentes se alcanza aproximadamente el 71.9% de varianza explicada.

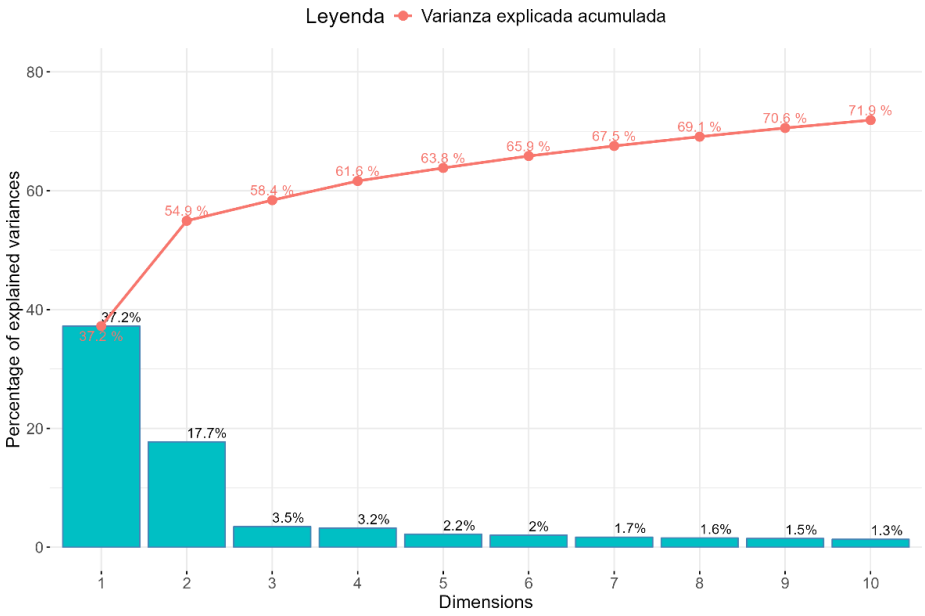


Figura 4. El gráfico muestra la distribución de la varianza explicada por cada componente principal en un análisis de componentes principales. Las barras azules representan la varianza explicada por cada componente individual, mientras que la línea roja acumulativa muestra el porcentaje total de varianza explicada a medida que se suman más componentes.

Adicionalmente, en la Figura 5 se presenta un diagrama de dispersión de los resultados de un análisis de componentes principales (PCA), que muestra la proyección de las muestras en las dos primeras componentes principales. Los puntos azules representan las muestras clasificadas como *Healthy,* y los puntos rojos, aquellas clasificadas como *SLE*.

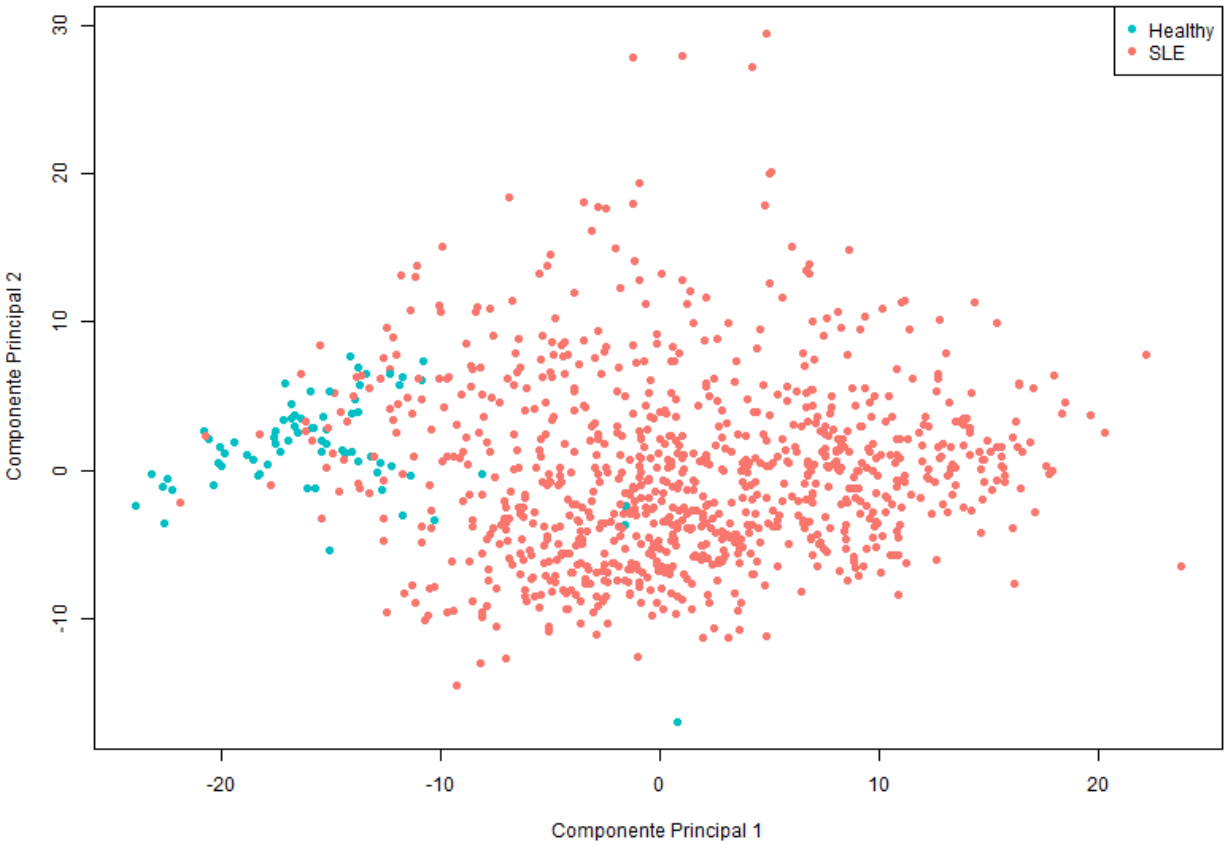


Figura 5. Gráfico de dispersión de los resultados de un análisis de componentes principales (PCA). La gráfica muestra la distribución de los datos proyectados sobre los dos primeros componentes principales obtenidos a partir de un PCA. Los puntos en color azul representan individuos sanos (*Healthy*), mientras que los puntos en color rojo representan individuos con lupus eritematoso sistémico (SLE).

Para complementar el análisis PCA se elaboró un mapa de calor que muestra la expresión media de los genes para cada clase. La Figura 6 presenta este mapa de calor, donde las diferentes tonalidades de rojo y verde indican los niveles de expresión promedio por clase, con rojo indicando menor expresión y verde mayor. El dendograma en la parte superior del gráfico indica grupos de genes que se expresan de manera similar, lo que puede reflejar rutas biológicas compartidas o respuestas coordinadas dentro de cada clase.

Figura 6. Mapa de calor de la expresión génica promedio por clase. El mapa de calor muestra los niveles de expresión génica promedio para dos clases: individuos sanos (*Healthy*) y aquellos con lupus eritematoso sistémico (SLE). Los colores en el mapa representan diferentes niveles de expresión génica, con tonos de verde indicando baja expresión y tonos de rojo indicando alta expresión. El dendrograma en la parte superior del mapa muestra la agrupación jerárquica de los genes según sus patrones de expresión.

Un análisis diferencial de la expresión genética se presenta en la Figura 7 a través de un *Volcano plot*, que muestra la relación entre la magnitud del cambio en la expresión (Log_2 *SLE/Healthy*) y la significancia estadística ($-Log_{10}$ *Valor-P*). Los puntos rojos indican genes sobreexpresados en *SLE*, y los azules en individuos sanos (*Healthy*). Este análisis revela que 88 genes están sobreexpresados en el grupo *Healthy* y 112 en el grupo *SLE*.

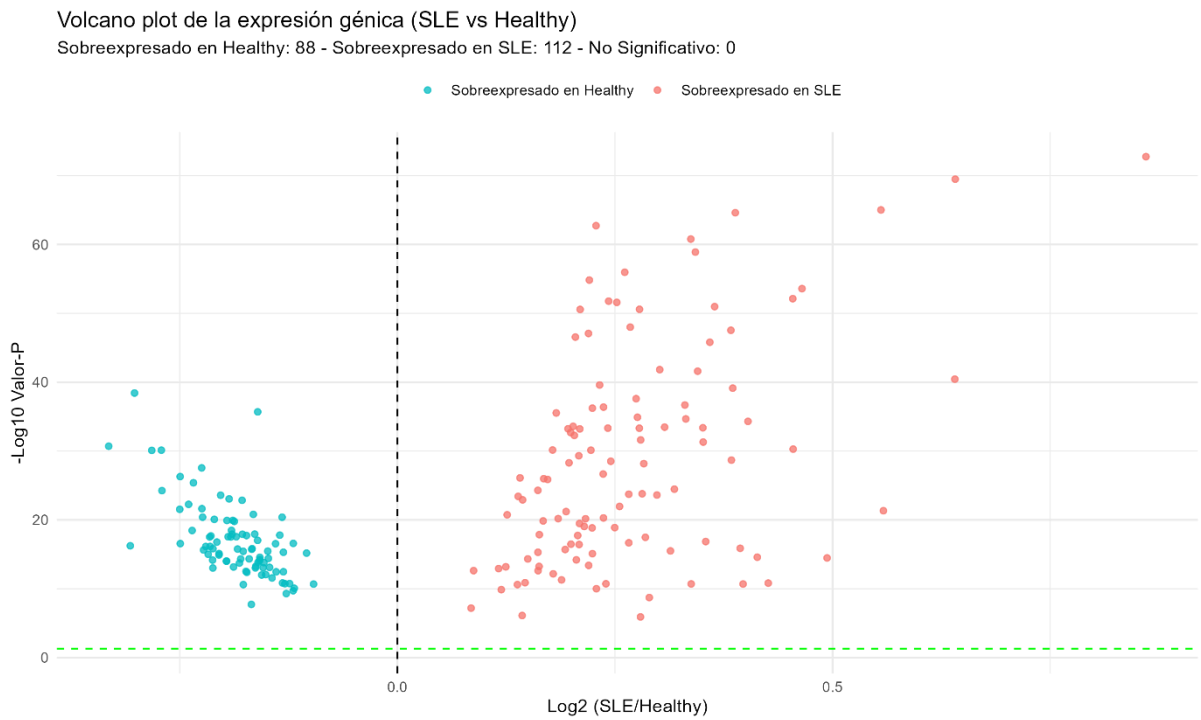


Figura 7. *Volcano plot*. El gráfico de dispersión muestra los resultados de un análisis de expresión génica diferencial. En el eje X se representa el logaritmo base 2 del cambio de expresión (*log_2 SLE/Healthy*), mientras que en el eje Y se muestra el valor negativo del logaritmo base 10 del p-valor *(-log_{10} valor-P*). Los puntos azules representan genes sobreexpresados en pacientes sanos, mientras que los puntos rojos indican genes sobreexpresados en pacientes con SLE. La línea vertical discontinua representa un umbral de cambio de expresión, y la línea horizontal verde indica un umbral de significancia estadística.

6.2. Metodología de análisis

Para llevar a cabo un análisis predictivo en el presente Trabajo de Fin de Máster, se ha diseñado una metodología estructurada en torno a cuatro pilares fundamentales: la selección de algoritmos, las técnicas de preprocesamiento de datos, la aplicación de validación cruzada estratificada y la elección de métricas de evaluación adecuadas. Para llevar a cabo el análisis hemos utilizado el lenguaje de programación R. El código empleado se puede consultar en la Sección de Anexos. A continuación, se describen en detalle cada uno de estos componentes.

Se han seleccionado tres algoritmos de minería de datos ampliamente utilizados por la comunidad científica que pertenecen a 3 paradigmas de clasificación diferentes. C4.5 (Quinlan, 1992), que genera árboles de decisión, *Random Forest* (Breiman, 2001) que consiste en un conjunto de árboles de decisión, lo que lo hace robusto frente a sobreajuste y SVM (Boser et al., 1992), que es un algoritmo conocido por su capacidad para manejar espacios de alta dimensión. Además, este es adecuado para datos donde las clases no son linealmente separables.

Para validar la capacidad de generalización de los modelos, se empleará la técnica de validación cruzada estratificada de 5 particiones (Berrar, 2018). Este enfoque divide el conjunto de datos en cinco partes iguales, utilizando secuencialmente cada una como conjunto de prueba mientras que las

restantes forman el conjunto de entrenamiento. La partición estratificada asegura que cada una de las 5 particiones mantenga la misma proporción de clases que el conjunto de datos original, lo que es especialmente útil cuando se trabaja con datos desequilibrados.

Las métricas de evaluación seleccionadas permiten una comparación entre los modelos. En este trabajo se han utilizado las siguientes métricas: la exactitud, que evalúa la proporción total de predicciones correctas; la precisión, que mide la proporción de identificaciones positivas que fueron realmente correctas; la sensibilidad, que indica cómo de efectivo es el modelo identificando positivos reales; la especificidad, que muestra cómo de efectivo es el modelo identificando negativos reales; la media geométrica entre sensibilidad y especificidad, que proporciona un balance entre la sensibilidad y la especificidad, útil en casos de clases desequilibradas; el *F1 score,* que proporciona un balance entre la precisión y la sensibilidad, útil en casos de clases desequilibradas; el área bajo la curva ROC, que mide la capacidad del modelo para discriminar entre las clases a diversos umbrales de decisión; finalmente, el coeficiente de correlación de Matthews, que ofrece una medida de calidad del modelo en términos de precisión y sensibilidad en todas las clases, considerada una de las métricas más informativas en problemas de clasificación binaria. Aunque se ha incorporado la exactitud y la precisión para hacer el estudio más exhaustivo, sus resultados no son fiables para determinar cuál técnica es más efectiva con datos desequilibrados como los analizados. Por ejemplo, se puede tener una alta exactitud en un clasificador que siempre predice la clase mayoritaria, o una precisión alta para un modelo que predice correctamente solo unas pocas muestras de la clase minoritaria. En ambos casos, las métricas no son representativas.

La Figura 8 ilustra en diagramas de flujo las estrategias de preprocesamiento y clasificación de datos utilizados en este trabajo.

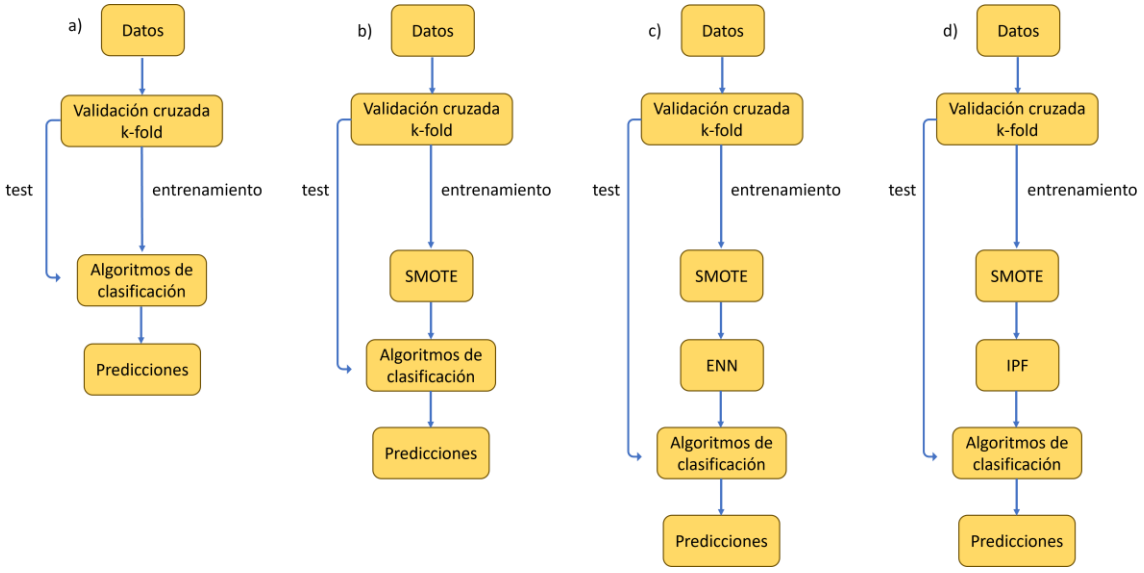


Figura 8. Diagramas de flujo de diferentes estrategias de preprocesamiento y clasificación. La Figura representa los cuatro enfoques utilizados para el preprocesamiento de datos y su clasificación: la Figura 8a incluye preprocesamiento, validación cruzada estratificada k-

fold, aplicación de algoritmos de clasificación y generación de predicciones; la Figura 8b añade el paso de *oversampling* con SMOTE después de la validación cruzada k-fold antes de aplicar los algoritmos de clasificación; la Figura 8c, después de SMOTE, incluye el método de eliminación de ruido con ENN antes de aplicar los algoritmos de clasificación; y la Figura 8d es similar al tercero, pero en lugar de ENN, se utiliza el filtro IPF después de SMOTE.

7. Resultados

En esta Sección, se analizan los resultados obtenidos de aplicar distintas técnicas de manejo de desequilibrio de clases sobre tres algoritmos de minería de datos y se comparan con el modelo inicial sin preprocesamiento. Los modelos evaluados son el modelo inicial sin preprocesamiento (None), el modelo incluyendo SMOTE (SMOTE), el modelo incluyendo SMOTE combinado con ENN (SMOTE+ENN) y el modelo incluyendo SMOTE combinado con IPF (SMOTE+IPF). Cada combinación ha sido analizada en términos de varias métricas de rendimiento: exactitud, precisión, sensibilidad, especificidad, media geométrica (GM), *F1 score*, área bajo la curva ROC y coeficiente de correlación de Matthews. Recordemos que aunque se ha incluido la exactitud y la precisión, los resultados de estas métricas no son fiables para determinar cuál técnica es más efectiva con los datos desequilibrados. Los resultados obtenidos de la experimentación se muestran en la Tabla 3.

Algoritmo	Métrica	None	SMOTE	SMOTE+ENN	SMOTE+IPF
C4.5	Exactitud	**0.9608**	**0.9608**	0.9428	0.9568
	Precisión	**0.7619**	0.7002	0.5731	0.6609
	Sensibilidad	0.6819	0.8229	**0.8505**	0.8352
	Especificidad	**0.9827**	0.9718	0.9502	0.9665
	GM	0.8136	0.8920	**0.8974**	0.8971
	F1 score	0.7088	**0.7520**	0.6839	0.7352
	AUC	0.7898	0.8811	**0.8912**	0.8882
	MCC	0.6952	**0.7365**	0.6697	0.7195
Random Forest	Exactitud	0.9709	**0.9739**	0.9618	0.9699
	Precisión	**0.8870**	0.8282	0.6825	0.7768
	Sensibilidad	0.6971	0.8210	**0.9038**	0.8210
	Especificidad	**0.9924**	0.9859	0.9665	0.9816
	GM	0.8294	0.8989	**0.9342**	0.8969
	F1 score	0.7727	**0.8204**	0.7751	0.7967
	AUC	**0.9870**	0.9858	0.9848	0.9856
	MCC	0.7682	**0.8088**	0.7652	0.7817
SVM	Exactitud	0.9679	**0.9769**	0.9448	0.9719
	Precisión	0.7862	**0.8078**	0.5735	0.7597
	Sensibilidad	0.7800	0.9048	**0.9457**	0.9181
	Especificidad	**0.9827**	**0.9827**	0.9448	0.9762
	GM	0.8738	0.9419	0.9449	**0.9460**
	F1 score	0.7776	**0.8497**	0.7124	0.8267
	AUC	0.9752	0.9859	0.9812	**0.9864**
	MCC	0.7635	**0.8411**	0.7111	0.8187

Tabla 3. Comparación de métricas de rendimiento entre diferentes modelos y técnicas de preprocesamiento. La Tabla presenta los resultados de exactitud, precisión, sensibilidad, especificidad, media geométrica (GM), *F1 score*, área bajo la curva ROC (AUC) y coeficiente de correlación de Matthews (MCC) para tres algoritmos de clasificación (C4.5, *Random Forest* y SVM), utilizando un modelo inicial sin

preprocesamiento (None), un modelo con SMOTE (SMOTE), un modelo con SMOTE y filtro ENN (SMOTE+ENN), y un modelo con SMOTE y filtro IPF (SMOTE+IPF). La exactitud y la precisión están resaltadas en rojo debido a que son métricas que no representan de manera adecuada el rendimiento de un modelo cuando se trabaja con datos desequilibrados. La sensibilidad y la especificidad se presentan en azul, ya que deben ser consideradas conjuntamente, dado que cada una se enfoca en describir una de las clases (minoritaria o mayoritaria) de los datos. GM, *F1 score*, AUC y MCC están resaltadas en verde, pues son las métricas que mejor describen el rendimiento del modelo. En negrita se resalta el valor más alto para cada combinación de algoritmo y métrica.

Teniendo en cuenta los resultados de la Tabla 3, el preprocesamiento demuestra ser eficaz para mejorar el rendimiento de los modelos en la clasificación no balanceada. Específicamente, la sensibilidad, crucial para detectar la clase minoritaria, muestra mejoras significativas en todos los algoritmos al aplicar SMOTE. Es interesante señalar que aunque los modelos sin preprocesar tienen una especificidad muy alta (predicen correctamente muchas muestras de las clases mayoritarias), la sensibilidad es muy baja en comparación con los modelos preprocesados. Sin embargo, el preprocesamiento mantiene valores similares de especificidad pero mejora significativamente la sensibilidad, demostrando los beneficios en el manejo de datos desequilibrados.

Para el algoritmo C4.5, aunque la exactitud se mantiene constante o mejora ligeramente con el preprocesamiento, es la sensibilidad la que muestra mejoras notables, particularmente con SMOTE y SMOTE+ENN. Esto indica una mejor capacidad del modelo para identificar correctamente la clase minoritaria. Sin embargo, la precisión disminuye con estos métodos, lo que sugiere un aumento en los falsos positivos. Las métricas más completas, como la media geométrica y el *F1 score*, también mejoran, sugiriendo que SMOTE y sus combinaciones ofrecen un balance más equilibrado entre precisión y sensibilidad.

El algoritmo *Random Forest* muestra una mejora en la sensibilidad con SMOTE y SMOTE+ENN, mientras que la precisión y especificidad se mantienen relativamente altas. La media geométrica y el MCC también muestran mejoras, destacando la eficacia de SMOTE para este algoritmo.

En el caso del algoritmo SVM, se observa que SMOTE y SMOTE+IPF mejoran tanto la sensibilidad como la media geométrica, reflejando una mejora en la capacidad del modelo para predecir correctamente la clase minoritaria y mantener un buen equilibrio entre las predicciones de ambas clases. Sin embargo, la combinación de SMOTE con ENN disminuye la precisión drásticamente debido al aumento de falsos positivos.

En este estudio, métricas como la media geométrica, el *F1 score*, el AUC y el MCC han demostrado ser más fiables que la exactitud para evaluar modelos en contextos de clasificación no balanceada. La exactitud puede ser engañosa en estos casos porque no refleja adecuadamente el rendimiento en la clase minoritaria. Entre estas métricas, el AUC y el MCC son especialmente útiles porque consideran tanto la sensibilidad como la especificidad, proporcionando una visión más completa del rendimiento del modelo. Cabe señalar que AUC es más utilizada en la evaluación de modelos porque además de

representar adecuadamente las clases mayoritaria y minoritaria, su interpretación es sencilla.

El único caso en el que el que el modelo sin preprocesamiento obtiene un mejor resultado es en el AUC con el algoritmo *Random Forest*. Sin embargo, todos los modelos presentan un rendimiento muy alto (aproximadamente 0.98), lo que deja un margen de mejora muy pequeño. Esta situación puede justificar por qué el modelo None sobresale en ese caso específico, ya que la alta base de rendimiento dificulta una posible mejora con las opciones de preprocesamiento.

Si se pone el foco en las métricas críticas para problemas de desequilibrio de clases, SVM con SMOTE generalmente obtiene las mejores métricas de rendimiento en media geométrica, *F1 score*, AUC y MCC por lo que sería la mejor opción para estudiar la base de datos estudiada. En contraste, el algoritmo C4.5 muestra el peor rendimiento para este conjunto de datos según estas mismas métricas.

Al comparar los filtros ENN e IPF se observa que ENN ha alcanzado la máxima puntuación en un mayor número de métricas. No obstante, en las métricas de media geométrica, *F1 score*, AUC y MCC, IPF demuestra ser superior. Los resultados indican que, especialmente para el algoritmo SVM, la combinación de SMOTE con IPF mejora significativamente estas métricas. Por otro lado, aunque ENN también contribuye a mejorar algunas métricas, su combinación con SMOTE puede llevar al aumento de falsos positivos.

Para finalizar esta Sección, podemos afirmar que los resultados evidencian que el preprocesamiento, especialmente mediante SMOTE, es efectivo para mejorar el rendimiento de los modelos en la clasificación no balanceada. No obstante, la combinación con filtros como ENN e IPF puede tener efectos variados, mejorando algunas métricas mientras deteriora otras, dependiendo del algoritmo. Sin preprocesamiento, los modelos tienden a obtener peores resultados en las métricas globales debido a la predominancia de la clase mayoritaria en los conjuntos de datos, lo que provoca un sesgo hacia esta y una baja sensibilidad para la clase minoritaria.

8. Conclusiones

En este Trabajo de Fin de Máster se ha analizado y evaluado la eficacia de técnicas de balanceo y filtrado de ruido aplicadas a la clasificación de datos transcriptómicos. Para ello, se ha aplicado y comparado la técnica de balanceo SMOTE y los métodos de filtrado de ruido ENN e IPF, en combinación con tres algoritmos de clasificación: C4.5, *Random Forest* y SVM. La eficacia de estas combinaciones se ha evaluado mediante diversas métricas de rendimiento, que han sido la exactitud, la precisión, la sensibilidad, la especificidad, la media geométrica, el *F1 score*, AUC y MCC, destacando las últimas cuatro por su relevancia en escenarios con datos desequilibrados.

A partir de los resultados obtenidos, se concluye que el preprocesamiento resulta eficaz para optimizar el rendimiento de los modelos en la clasificación de datos desequilibrados, ya que los modelos sin preprocesamiento obtienen resultados inferiores en las métricas globales debido a la dominancia de la clase mayoritaria en los datos. En este contexto, SMOTE se muestra generalmente efectivo para mejorar las métricas, especialmente en modelos como C4.5 y SVM. Los resultados señalan que la media geométrica, el *F1 score*, el AUC y el MCC son métricas más adecuadas que la exactitud o la precisión para evaluar el rendimiento en la clasificación no balanceada. Además, es crucial evaluar conjuntamente la sensibilidad y la especificidad, dado que cada una se enfoca en diferentes aspectos de los datos, específicamente en las clases minoritaria y mayoritaria, respectivamente. Entre los algoritmos analizados, SVM ha obtenido los mejores resultados, destacando el preprocesamiento de SVM con SMOTE como el modelo con mejores métricas de evaluación. Asimismo, el filtro IPF supera a ENN en métricas como la media geométrica, el *F1 score*, el AUC y el MCC.

Para continuar investigando este tema en trabajos futuros, se podrían explorar nuevas técnicas de balanceo de datos, como ADASYN y Borderline-SMOTE, así como filtros de ruido como *ClusterCentroids* y *CondensedNearestNeighbor*. Además, sería interesante estudiar diferentes combinaciones de estas técnicas para optimizar su eficacia. También sería útil evaluar algoritmos de clasificación avanzados como redes neuronales profundas y modelos *ensemble* como XGBoost y LightGBM. Asimismo, aplicar estas técnicas a conjuntos de datos transcriptómicos diversos ayudaría a verificar la capacidad de generalización de los resultados. Por último, resultaría interesante evaluar y optimizar los costos computacionales de las diferentes técnicas para mejorar su escalabilidad en grandes conjuntos de datos.

Bibliografía

Bello, N. M., & Renter, D. G. (2018). Invited review: Reproducible research from noisy data: Revisiting key statistical principles for the animal sciences. *Journal of Dairy Science*, *101*(7), 5679–5701. https://doi.org/https://doi.org/10.3168/jds.2017-13978

Berrar, D. (2018). Cross-validation. In *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics* (Vols. 1–3, pp. 542–545). Elsevier. https://doi.org/10.1016/B978-0-12-809633-8.20349-X

Bhavsar, H., & Panchal, M. H. (2012). A Review on Support Vector Machine for Data Classification. In *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* (Vol. 1, Issue 10).

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 144–152. https://doi.org/10.1145/130385.130401

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Chawla, N. V, Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. In *Journal of Artificial Intelligence Research* (Vol. 16).

Chen, X., Kang, Q., Zhou, M., & Wei, Z. (2016). A novel under-sampling algorithm based on Iterative-Partitioning Filters for imbalanced classification. *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, 490–494. https://doi.org/10.1109/COASE.2016.7743445

Das, S., Dey, A., & Roy, N. (2015). Applications of Artificial Intelligence in Machine Learning: Review and Prospect. In *International Journal of Computer Applications* (Vol. 115, Issue 9).

Donoho, D. L. (2000). *High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality*.

Elmaizi, A., Sarhrouni, E., Hammouch, A., & Chafik, N. (2022). *Hyperspectral Images Classification and Dimensionality Reduction using spectral interaction and SVM classifier*.

Fernández, A., del Río, S., Chawla, N. V, & Herrera, F. (2017). An insight into imbalanced Big Data classification: outcomes and challenges. *Complex & Intelligent Systems*, *3*(2), 105–120. https://doi.org/10.1007/s40747-017-0037-9

Fernández, A., García, S., Herrera, F., & Chawla, N. V. (2018). SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary. In *Journal of Artificial Intelligence Research* (Vol. 61).

Filippakis, P., Ougiaroglou, S., & Evangelidis, G. (2023). Condensed Nearest Neighbour Rules for Multi-Label Datasets. *Proceedings of the 27th International Database Engineered Applications Symposium*, 43–50. https://doi.org/10.1145/3589462.3589492

Fonseca, J., & Bacao, F. (2023). Geometric SMOTE for imbalanced datasets with nominal and continuous features. *Expert Systems with Applications*, *234*, 121053. https://doi.org/https://doi.org/10.1016/j.eswa.2023.121053

Fonseca, J., Douzas, G., & Bacao, F. (2021). Improving Imbalanced Land Cover Classification with K-Means SMOTE: Detecting and Oversampling Distinctive Minority Spectral Signatures. *Information*, *12*(7). https://doi.org/10.3390/info12070266

Gastwirth, J. L. (1972). The Estimation of the Lorenz Curve and Gini Index. *The Review of Economics and Statistics*, *54*(3), 306–316. https://doi.org/10.2307/1937992

Ghosh, K., Bellinger, C., Corizzo, R., Branco, P., Krawczyk, B., & Japkowicz, N. (2022). The class imbalance problem in deep learning. *Machine Learning*. https://doi.org/10.1007/s10994-022-06268-8

Gosain, A., & Sardana, S. (2017). Handling class imbalance problem using oversampling techniques: A review. *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 79–85. https://doi.org/10.1109/ICACCI.2017.8125820

Gotoh, J., & Uryasev, S. (2017). Support vector machines based on convex risk functions and general norms. *Annals of Operations Research*, *249*(1), 301–328. https://doi.org/10.1007/s10479-016-2326-x

Gupta, S., & Gupta, A. (2019). Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, *161*, 466–474. https://doi.org/10.1016/j.procs.2019.11.146

Guzmán-Ponce, A., Valdovinos, R. M., Sánchez, J. S., & Marcial-Romero, J. R. (2020). A new under-sampling method to face class overlap and imbalance. *Applied Sciences (Switzerland)*, *10*(15). https://doi.org/10.3390/app10155164

Han, H., Wang, W.-Y., & Mao, B.-H. (2005). Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In D.-S. Huang, X.-P. Zhang, & G.-B. Huang (Eds.), *Advances in Intelligent Computing* (pp. 878–887). Springer Berlin Heidelberg.

Hartmann, E. (1999). On the curvature of curves and surfaces defined by normalforms. *Computer Aided Geometric Design*, *16*(5), 355–376. https://doi.org/https://doi.org/10.1016/S0167-8396(99)00003-5

He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 1322–1328. https://doi.org/10.1109/IJCNN.2008.4633969

Hong, C. K. Y., Ramu, A., Zhao, S., & Cohen, B. A. (2022). Effect of genomic and cellular environments on gene expression noise. *BioRxiv*, 2022.08.31.506082. https://doi.org/10.1101/2022.08.31.506082

Hossin, M., & M.N, S. M. N. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), 01–11. https://doi.org/10.5121/ijdkp.2015.5201

Hulse, J. Van, Khoshgoftaar, T. M., & Napolitano, A. (2009). An empirical comparison of repetitive undersampling techniques. *2009 IEEE International Conference on Information Reuse & Integration*, 29–34. https://doi.org/10.1109/IRI.2009.5211614

Jiang, T., Gradus, J. L., & Rosellini, A. J. (2020). Supervised Machine Learning: A Brief Primer. *Behavior Therapy*, *51*(5), 675–687. https://doi.org/https://doi.org/10.1016/j.beth.2020.05.002

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, *6*(1), 27. https://doi.org/10.1186/s40537-019-0192-5

Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. In *Informatica*

(Vol. 31).

Krishnaiah, V., Narsimha, G., & Chandra, N. S. (2014). *Survey of Classification Techniques in Data Mining*. www.ijcaonline.org

Lemon, S. C., Roy, J., Clark, M. A., Friedmann, P. D., & Rakowski, W. (2003). Classification and regression tree analysis in public health: Methodological review and comparison with logistic regression. *Annals of Behavioral Medicine*, *26*(3), 172–181. https://doi.org/10.1207/S15324796ABM2603_02

Liang, X., Liu, X., & Yao, L. (2022). Review–A Survey of Learning from Noisy Labels. *ECS Sensors Plus*, *1*(2), 021401. https://doi.org/10.1149/2754-2726/ac75f5

Loh, W.-Y. (2011). Classification and regression trees. *WIREs Data Mining and Knowledge Discovery*, *1*(1), 14–23. https://doi.org/https://doi.org/10.1002/widm.8

Mohamed, W. N. H. W., Salleh, M. N. M., & Omar, A. H. (2012). A comparative study of Reduced Error Pruning method in decision tree algorithms. *2012 IEEE International Conference on Control System, Computing and Engineering*, 392–397. https://doi.org/10.1109/ICCSCE.2012.6487177

Mohammed, R., Rawashdeh, J., & Abdullah, M. (2020). Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. *2020 11th International Conference on Information and Communication Systems (ICICS)*, 243–248. https://doi.org/10.1109/ICICS49469.2020.239556

Morales, J. A., Saldaña, R., Santana-Castolo, M. H., Torres-Cerna, C. E., Borrayo, E., Mendizabal-Ruiz, A. P., Vélez-Pérez, H. A., & Mendizabal-Ruiz, G. (2020). Deep Learning for the Classification of Genomic Signals. *Mathematical Problems in Engineering*, *2020*. https://doi.org/10.1155/2020/7698590

Moutsopoulos, I., Maischak, L., Lauzikaite, E., Vasquez Urbina, S. A., Williams, E. C., Drost, H.-G., & Mohorianu, I. I. (2021). noisyR: enhancing biological signal in sequencing datasets by characterizing random technical noise. *Nucleic Acids Research*, *49*(14), e83–e83. https://doi.org/10.1093/nar/gkab433

Mukherjee, M., & Khushi, M. (2021). SMOTE-ENC: A Novel SMOTE-Based Method to Generate Synthetic Data for Nominal and Continuous Features. *Applied System Innovation*, *4*(1). https://doi.org/10.3390/asi4010018

Nasteski, V. (2017). An overview of the supervised machine learning methods. *HORIZONS.B*, *4*, 51–62. https://doi.org/10.20544/horizons.b.04.1.17.p05

Nettleton, D. F., Orriols-Puig, A., & Fornells, A. (2010). A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, *33*(4), 275–306. https://doi.org/10.1007/s10462-010-9156-z

Pradipta, G. A., Wardoyo, R., Musdholifah, A., Sanjaya, I. N. H., & Ismail, M. (2021). SMOTE for Handling Imbalanced Data Problem : A Review. *2021 Sixth International Conference on Informatics and Computing (ICIC)*, 1–8. https://doi.org/10.1109/ICIC54025.2021.9632912

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. https://doi.org/10.1007/BF00116251

Quinlan, J. R. (1992). *C4.5 Programs for Machine Learning*. Morgan Kaufmann.

Rajender, N., & Gopalachari, M. V. (2023). An efficient dimensionality reduction based on adaptive-GSM and

transformer assisted classification for high dimensional data. *International Journal of Information Technology*. https://doi.org/10.1007/s41870-023-01552-9

Ray, P., Reddy, S. S., & Banerjee, T. (2021). Various dimension reduction techniques for high dimensional data analysis: a review. *Artificial Intelligence Review*, *54*(5), 3473–3515. https://doi.org/10.1007/s10462-020-09928-0

Sáez, J. A., Krawczyk, B., & Woźniak, M. (2016). On the Influence of Class Noise in Medical Data Classification: Treatment Using Noise Filtering Methods. *Applied Artificial Intelligence*, *30*(6), 590–609. https://doi.org/10.1080/08839514.2016.1193719

Sáez, J. A., Luengo, J., Stefanowski, J., & Herrera, F. (2015). SMOTE-IPF: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences*, *291*(C), 184–203. https://doi.org/10.1016/j.ins.2014.08.051

Salcedo-Sanz, S., Rojo-Álvarez, J. L., Martínez-Ramón, M., & Camps-Valls, G. (2014). Support vector machines in engineering: an overview. *WIREs Data Mining and Knowledge Discovery*, *4*(3), 234–267. https://doi.org/https://doi.org/10.1002/widm.1125

Salzberg, S. L. (1994). C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning*, *16*(3), 235–240. https://doi.org/10.1007/BF00993309

Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Learning Local Feature Descriptors Using Convex Optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(8), 1573–1585. https://doi.org/10.1109/TPAMI.2014.2301163

Singh, A., Thakur, N., & Sharma, A. (2016). A review of supervised machine learning algorithms. *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 1310–1315.

Singh, H., Guru, S., & Sahib, G. (2013). New Method for Finding Initial Cluster Centroids in K-means Algorithm. In *International Journal of Computer Applications* (Vol. 74, Issue 6).

Tang, M., Liu, Y., & Gong, X. (2023). Multi-Omics Data Mining Techniques: Algorithms and Software. In K. Ning (Ed.), *Methodologies of Multi-Omics Data Integration and Data Mining: Techniques and Applications* (pp. 55–74). Springer Nature Singapore. https://doi.org/10.1007/978-981-19-8210-1_4

Tang, Y., Zhang, Y.-Q., Chawla, N. V, & Krasser, S. (2009). SVMs Modeling for Highly Imbalanced Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(1), 281–288. https://doi.org/10.1109/TSMCB.2008.2002909

Tasci, E., Zhuge, Y., Camphausen, K., & Krauze, A. V. (2022). Bias and Class Imbalance in Oncologic Data—Towards Inclusive and Transferrable AI in Large Scale Oncology Data Sets. In *Cancers* (Vol. 14, Issue 12). MDPI. https://doi.org/10.3390/cancers14122897

Vukadinovic, M., Renjith, G., Yuan, V., Kwan, A., Cheng, S. C., Li, D., Clarke, S. L., & Ouyang, D. (2024). *Impact of Measurement Noise on Genetic Association Studies of Cardiac Function*. 134.

Wilson, D. L. (1972). Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Trans. Syst. Man Cybern.*, *2*, 408–421. https://api.semanticscholar.org/CorpusID:6699477

Xie, C., Niu, Y., Ping, J., Wang, Y., Yang, C., Li, Y., & Zhou, G. (2021). Genome-wide association study identifies new loci associated with noise-induced tinnitus in Chinese populations. *BMC Genomic Data*, *22*(1), 31.

https://doi.org/10.1186/s12863-021-00987-y

Xiong, H., Pandey, G., Steinbach, M., & Kumar, V. (2006). Enhancing data analysis with noise removal. *IEEE Transactions on Knowledge and Data Engineering*, *18*(3), 304–319. https://doi.org/10.1109/TKDE.2006.46

Anexos

A.1. Código R utilizado en el análisis de datos en el modelo None

```
library(caret)
library(e1071)
library(pROC)
library(MLmetrics)

# Cargar datos
gendata <- read.table("gendata.csv", header = TRUE, sep = ";")
gendata$class <- as.factor(gendata$class)

# Configurar la validación cruzada
set.seed(123)
folds <- createFolds(gendata$class, k = 5, list = TRUE, returnTrain = TRUE)

# Preparar para almacenar modelos y métricas de rendimiento para cada
algoritmo
models_list <- list(svm_models = list(), rf_models = list(), j48_models =
list())
performance_metrics <- list(svm = list(), rf = list(), j48 = list())

# Bucle a través de cada pliegue
for(i in seq_along(folds)) {
  # Dividir los datos en conjuntos de entrenamiento y prueba
  trainData <- gendata[folds[[i]], ]
  testData <- gendata[-folds[[i]], ]

  # Configuración común de trainControl para todos los modelos
  fitControl <- trainControl(
    method = "none",
    savePredictions = "final",
    classProbs = TRUE,
    summaryFunction = twoClassSummary
  )

  # Entrenar modelo SVM
  svm_model <- train(class ~ ., data = trainData, method = "svmRadial",
                     trControl = fitControl, preProcess = "scale", metric =
"ROC")
  models_list$svm_models[[i]] <- svm_model

  # Entrenar modelo Random Forest
  rf_model <- train(class ~ ., data = trainData, method = "rf",
                    trControl = fitControl, metric = "ROC")
  models_list$rf_models[[i]] <- rf_model

  # Entrenar modelo J48
  j48_model <- train(class ~ ., data = trainData, method = "J48",
                     trControl = fitControl, metric = "ROC")
  models_list$j48_models[[i]] <- j48_model

  # Evaluar el rendimiento del modelo en el conjunto de prueba para cada
modelo
  for(model_type in names(models_list)) {
    probabilities <- predict(models_list[[model_type]][[i]], testData, type
= "prob")
```

```
    predicted_classes <- predict(models_list[[model_type]][[i]], testData,
type = "raw")
    actual_classes <- testData$class

    cm <- confusionMatrix(data = predicted_classes, reference =
actual_classes)

    # Cálculo personalizado de MCC
    tp <- as.numeric(cm$table[1, 1])  # Verdaderos Positivos
    tn <- as.numeric(cm$table[2, 2])  # Verdaderos Negativos
    fp <- as.numeric(cm$table[1, 2])  # Falsos Positivos
    fn <- as.numeric(cm$table[2, 1])  # Falsos Negativos
    denom = sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
    if (denom == 0) {
      mcc = 0
    } else {
      mcc = (tp * tn - fp * fn) / denom
    }

    roc <- roc(response = actual_classes, predictor = probabilities[,2])
    auc_value <- roc$auc

    accuracy <- (tp+tn)/(tp+tn+fp+fn)
    sensitivity <- cm$byClass['Sensitivity']
    specificity <- cm$byClass['Specificity']
    precision <- cm$byClass['Precision']
    f1 <- cm$byClass['F1']
    geom_average = sqrt(sensitivity*specificity)

    performance_metrics[[model_type]][[i]] <- list(
      AUC = auc_value,
      Accuracy = accuracy,
      Sensitivity = sensitivity,
      Specificity = specificity,
      F1 = f1,
      MCC = mcc,
      Geom_average = geom_average,
      Precision = precision
    )
  }
}

# Imprimir métricas de rendimiento para revisión
print(performance_metrics)

# Inicializar una lista para almacenar las métricas promedio para cada
modelo
average_metrics <- list(svm = list(), rf = list(), j48 = list())

# Calcular el promedio de cada métrica para los modelos SVM
for(metric in c("Accuracy", "Precision", "Sensitivity", "Specificity",
"Geom_average", "F1", "AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$svm, function(x) x[[metric]])
  average_metrics$svm[[metric]] <- mean(metric_values)
}

# Calcular el promedio de cada métrica para los modelos RF
for(metric in c("Accuracy", "Precision", "Sensitivity", "Specificity",
"Geom_average", "F1", "AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$rf, function(x) x[[metric]])
  average_metrics$rf[[metric]] <- mean(metric_values)
```

```
}

# Calcular el promedio de cada métrica para los modelos J48
for(metric in c("Accuracy", "Precision", "Sensitivity", "Specificity",
"Geom_average", "F1", "AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$j48, function(x) x[[metric]])
  average_metrics$j48[[metric]] <- mean(metric_values)
}

# Imprimir las métricas promedio para cada modelo
print(average_metrics)

# Empezar con un marco de datos vacío para almacenar los resultados
long_df_initial <- data.frame(Algorithm = character(),
                              Metric = character(),
                              Average_initial = numeric(),
                              stringsAsFactors = FALSE)

# Bucle a través de cada algoritmo
for(alg in names(average_metrics)) {
  # Para cada algoritmo, crear un marco de datos temporal de sus métricas
  temp_df <- data.frame(Algorithm = alg,
                        Metric = names(average_metrics[[alg]]),
                        Average_initial = unlist(average_metrics[[alg]]),
                        stringsAsFactors = FALSE)

  # Unir el marco de datos temporal al marco de datos largo
  long_df_initial <- rbind(long_df_initial, temp_df)
}

# Eliminar nombres de filas para asegurar que no se impriman
rownames(long_df_initial) <- NULL

# Imprimir el marco de datos limpio en formato largo
print(long_df_initial)
```

A.2. Código R utilizado en el análisis de datos en el modelo con SMOTE

```
# Cargar las bibliotecas requeridas
library(caret)
library(DMwR2)
library(randomForest)
library(e1071)
library(RWeka)
library(pROC)
library(caret)
library(smotefamily)
library(NoiseFiltersR)
library(e1071)
library(pROC)
library(MLmetrics)

# Cargar los datos
gendata <- read.table("gendata.csv", header = TRUE, sep = ";")
gendata$class <- as.factor(gendata$class)

# Configurar la validación cruzada
set.seed(123)  # para reproducibilidad
folds <- createFolds(gendata$class, k = 5, list = TRUE, returnTrain = TRUE)
```

```
# Preparar para almacenar modelos y métricas de rendimiento para cada
algoritmo
models_list <- list(svm_models = list(), rf_models = list(), j48_models =
list())
performance_metrics <- list(svm = list(), rf = list(), j48 = list())

# Bucle a través de cada fold
for(i in seq_along(folds)) {
  # Dividir los datos en conjuntos de entrenamiento y prueba
  trainData <- gendata[folds[[i]], ]
  testData <- gendata[-folds[[i]], ]

  # Aplicar SMOTE a los datos de entrenamiento
  syntheticData <- smotefamily::SMOTE(X = trainData[,-ncol(trainData)],
                                      target = trainData[,ncol(trainData)],
                                      K = 5)$syn_data

  # Combinar los datos de entrenamiento originales con los datos sintéticos
generados por SMOTE
  combinedData <- rbind(trainData, syntheticData)
  combinedData$class <- as.factor(combinedData$class)

  # No aplicar IPF al conjunto de datos combinado
  filteredData <- combinedData

  # Configuración común de trainControl para todos los modelos
  fitControl <- trainControl(
    method = "none",
    savePredictions = "final",
    classProbs = TRUE,
    summaryFunction = twoClassSummary
  )

  # Entrenar el modelo SVM
  svm_model <- train(class ~ ., data = filteredData, method = "svmRadial",
                     trControl = fitControl, preProcess = "scale", metric =
"ROC")
  models_list$svm_models[[i]] <- svm_model

  # Entrenar el modelo Random Forest
  rf_model <- train(class ~ ., data = filteredData, method = "rf",
                    trControl = fitControl, metric = "ROC")
  models_list$rf_models[[i]] <- rf_model

  # Entrenar el modelo J48
  j48_model <- train(class ~ ., data = filteredData, method = "J48",
                     trControl = fitControl, metric = "ROC")
  models_list$j48_models[[i]] <- j48_model

  # Evaluar el rendimiento del modelo en el conjunto de prueba para cada
modelo
  for(model_type in names(models_list)) {
    probabilities <- predict(models_list[[model_type]][[i]], testData, type
= "prob")
    predicted_classes <- predict(models_list[[model_type]][[i]], testData,
type = "raw")
    actual_classes <- testData$class
    cm <- confusionMatrix(data = predicted_classes, reference =
actual_classes)

    # Cálculo personalizado de MCC
```

```
    tp <- as.numeric(cm$table[1, 1])  # Verdaderos positivos
    tn <- as.numeric(cm$table[2, 2])  # Verdaderos negativos
    fp <- as.numeric(cm$table[1, 2])  # Falsos positivos
    fn <- as.numeric(cm$table[2, 1])  # Falsos negativos
    denom = sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
    if (denom == 0) {
      mcc = 0  # Prevenir división por cero
    } else {
      mcc = (tp * tn - fp * fn) / denom
    }
    roc <- roc(response = actual_classes, predictor = probabilities[,2])
    auc_value <- roc$auc

    accuracy <- (tp+tn)/(tp+tn+fp+fn)
    sensitivity <- cm$byClass['Sensitivity']
    specificity <- cm$byClass['Specificity']
    #recall <- tp/(tp+tn)
    precision <- cm$byClass['Precision']
    f1 <- cm$byClass['F1']
    geom_average = sqrt(sensitivity*specificity)

    performance_metrics[[model_type]][[i]] <- list(
      AUC = auc_value,
      Accuracy = accuracy,
      Sensitivity = sensitivity,
      Specificity = specificity,
      F1 = f1,
      MCC = mcc,
      #Recall = recall,
      Geom_average = geom_average,
      Precision = precision
    )
  }
}

print(performance_metrics)

# Inicializar una lista para almacenar las métricas promedio para cada
modelo
average_metrics <- list(svm = list(), rf = list(), j48 = list())

# Calcular el promedio de cada métrica para los modelos SVM
for(metric in c( "Accuracy","Precision",  "Sensitivity", "Specificity",
"Geom_average", "F1","AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$svm_models, function(x)
x[[metric]])
  average_metrics$svm[[metric]] <- mean(metric_values)
}

# Calcular el promedio de cada métrica para los modelos RF
for(metric in c( "Accuracy","Precision",  "Sensitivity", "Specificity",
"Geom_average", "F1","AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$rf_models, function(x)
x[[metric]])
  average_metrics$rf[[metric]] <- mean(metric_values)
}

# Calcular el promedio de cada métrica para los modelos J48
for(metric in c( "Accuracy", "Precision", "Sensitivity", "Specificity",
"Geom_average", "F1","AUC", "MCC")) {
```

```
  metric_values <- sapply(performance_metrics$j48_models, function(x)
x[[metric]])
  average_metrics$j48[[metric]] <- mean(metric_values)
}

# Imprimir las métricas promedio para cada modelo
print(average_metrics)

# Comenzar con un data frame vacío para contener los resultados
long_df_smote <- data.frame(Algorithm = character(),
                            Metric = character(),
                            Average_smote = numeric(),
                            stringsAsFactors = FALSE)

# Bucle a través de cada algoritmo
for(alg in names(average_metrics)) {
  # Para cada algoritmo, crear un data frame temporal de sus métricas
  temp_df <- data.frame(Algorithm = alg,
                        Metric = names(average_metrics[[alg]]),
                        Average_smote = unlist(average_metrics[[alg]]),
                        stringsAsFactors = FALSE)

  # Unir el data frame temporal al data frame largo
  long_df_smote <- rbind(long_df_smote, temp_df)
}

# Eliminar nombres de filas para asegurar que no se impriman
rownames(long_df_smote) <- NULL

# Imprimir el data frame limpio en formato largo
print(long_df_smote)
```

A.3. Código R utilizado en el análisis de datos en el modelo con SMOTE combinado con ENN

```
library(caret)
library(smotefamily)
library(NoiseFiltersR)
library(e1071)
library(pROC)
library(MLmetrics)

# Cargar datos
gendata <- read.table("gendata.csv", header = TRUE, sep = ";")
gendata$class <- as.factor(gendata$class)

# Configurar validación cruzada
set.seed(123)  # para reproducibilidad
folds <- createFolds(gendata$class, k = 5, list = TRUE, returnTrain = TRUE)

# Preparar para almacenar modelos y métricas de rendimiento para cada
algoritmo
models_list <- list(svm_models = list(), rf_models = list(), j48_models =
list())
performance_metrics <- list(svm = list(), rf = list(), j48 = list())

# Bucle a través de cada fold
for(i in seq_along(folds)) {
  # Dividir datos en conjuntos de entrenamiento y prueba
  trainData <- gendata[folds[[i]], ]
```

```
  testData <- gendata[-folds[[i]], ]

  # Aplicar SMOTE a los datos de entrenamiento
  syntheticData <- smotefamily::SMOTE(X = trainData[,-ncol(trainData)],
                                      target = trainData[,ncol(trainData)],
                                      K = 5)$syn_data

  # Combinar datos de entrenamiento originales con datos sintéticos
generados por SMOTE
  combinedData <- rbind(trainData, syntheticData)
  combinedData$class <- as.factor(combinedData$class)

  # Aplicar ENN al conjunto de datos combinado
  filteredData <- ENN(combinedData, classColumn =
ncol(combinedData))$cleanData

  # Configuración común de trainControl para todos los modelos
  fitControl <- trainControl(
    method = "none",
    savePredictions = "final",
    classProbs = TRUE,
    summaryFunction = twoClassSummary
  )

  # Entrenar modelo SVM
  svm_model <- train(class ~ ., data = filteredData, method = "svmRadial",
                     trControl = fitControl, preProcess = "scale", metric =
"ROC")
  models_list$svm_models[[i]] <- svm_model

  # Entrenar modelo Random Forest
  rf_model <- train(class ~ ., data = filteredData, method = "rf",
                    trControl = fitControl, metric = "ROC")
  models_list$rf_models[[i]] <- rf_model

  # Entrenar modelo J48
  j48_model <- train(class ~ ., data = filteredData, method = "J48",
                     trControl = fitControl, metric = "ROC")
  models_list$j48_models[[i]] <- j48_model

  # Evaluar el rendimiento del modelo en el conjunto de prueba para cada
modelo
  for(model_type in names(models_list)) {
    probabilities <- predict(models_list[[model_type]][[i]], testData, type
= "prob")
    predicted_classes <- predict(models_list[[model_type]][[i]], testData,
type = "raw")
    actual_classes <- testData$class
    cm <- confusionMatrix(data = predicted_classes, reference =
actual_classes)

    # Cálculo personalizado de MCC
    tp <- as.numeric(cm$table[1, 1])  # Verdaderos Positivos
    tn <- as.numeric(cm$table[2, 2])  # Verdaderos Negativos
    fp <- as.numeric(cm$table[1, 2])  # Falsos Positivos
    fn <- as.numeric(cm$table[2, 1])  # Falsos Negativos
    denom = sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
    if (denom == 0) {
      mcc = 0
    } else {
      mcc = (tp * tn - fp * fn) / denom
```

```
    }
    roc <- roc(response = actual_classes, predictor = probabilities[,2])
    auc_value <- roc$auc

    accuracy <- (tp + tn) / (tp + tn + fp + fn)
    sensitivity <- cm$byClass['Sensitivity']
    specificity <- cm$byClass['Specificity']
    precision <- cm$byClass['Precision']
    f1 <- cm$byClass['F1']
    geom_average = sqrt(sensitivity * specificity)

    performance_metrics[[model_type]][[i]] <- list(
      AUC = auc_value,
      Accuracy = accuracy,
      Sensitivity = sensitivity,
      Specificity = specificity,
      F1 = f1,
      MCC = mcc,
      Geom_average = geom_average,
      Precision = precision
    )
  }
}

print(performance_metrics)

# Inicializar una lista para almacenar las métricas promedio para cada
modelo
average_metrics <- list(svm = list(), rf = list(), j48 = list())

# Calcular el promedio de cada métrica para los modelos SVM
for(metric in c("Accuracy", "Precision", "Sensitivity", "Specificity",
"Geom_average", "F1", "AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$svm, function(x) x[[metric]])
  average_metrics$svm[[metric]] <- mean(metric_values)
}

# Calcular el promedio de cada métrica para los modelos RF
for(metric in c("Accuracy", "Precision", "Sensitivity", "Specificity",
"Geom_average", "F1", "AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$rf, function(x) x[[metric]])
  average_metrics$rf[[metric]] <- mean(metric_values)
}

# Calcular el promedio de cada métrica para los modelos J48
for(metric in c("Accuracy", "Precision", "Sensitivity", "Specificity",
"Geom_average", "F1", "AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$j48, function(x) x[[metric]])
  average_metrics$j48[[metric]] <- mean(metric_values)
}

# Imprimir las métricas promedio para cada modelo
print(average_metrics)

# Comenzar con un data frame vacío para contener los resultados
long_df_smote_enn <- data.frame(Algorithm = character(),
                                Metric = character(),
                                Average_smote_enn = numeric(),
                                stringsAsFactors = FALSE)

# Bucle a través de cada algoritmo
```

```
for(alg in names(average_metrics)) {
  # Para cada algoritmo, crear un data frame temporal de sus métricas
  temp_df <- data.frame(Algorithm = alg,
                        Metric = names(average_metrics[[alg]]),
                        Average_smote_enn = unlist(average_metrics[[alg]]),
                        stringsAsFactors = FALSE)

  # Unir el data frame temporal al data frame largo
  long_df_smote_enn <- rbind(long_df_smote_enn, temp_df)
}

# Eliminar nombres de fila para asegurar que no se impriman
rownames(long_df_smote_enn) <- NULL

# Imprimir el data frame en formato largo limpio
print(long_df_smote_enn)
```

A.4. Código R utilizado en el análisis de datos en el modelo con SMOTE combinado con IPF

```
library(caret)
library(smotefamily)
library(NoiseFiltersR)
library(e1071)
library(pROC)
library(MLmetrics)

# Cargar los datos
gendata <- read.table("gendata.csv", header = TRUE, sep = ";")
gendata$class <- as.factor(gendata$class)

# Configurar la validación cruzada
set.seed(123)  # para reproducibilidad
folds <- createFolds(gendata$class, k = 5, list = TRUE, returnTrain = TRUE)

# Preparar para almacenar modelos y métricas de rendimiento para cada
algoritmo
models_list <- list(svm_models = list(), rf_models = list(), j48_models =
list())
performance_metrics <- list(svm = list(), rf = list(), j48 = list())

# Bucle a través de cada pliegue
for(i in seq_along(folds)) {
  # Dividir los datos en conjuntos de entrenamiento y prueba
  trainData <- gendata[folds[[i]], ]
  testData <- gendata[-folds[[i]], ]

  # Aplicar SMOTE a los datos de entrenamiento
  syntheticData <- smotefamily::SMOTE(X = trainData[,-ncol(trainData)],
                                      target = trainData[,ncol(trainData)],
                                      K = 5)$syn_data

  # Combinar los datos de entrenamiento originales con los datos sintéticos
generados por SMOTE
  combinedData <- rbind(trainData, syntheticData)
  combinedData$class <- as.factor(combinedData$class)

  # Aplicar IPF al conjunto de datos combinado
  filteredData <- IPF(combinedData, classColumn = ncol(combinedData),
nfolds = 5)[["cleanData"]]
```

```
  # Configuración común de trainControl para todos los modelos
  fitControl <- trainControl(
    method = "none",
    savePredictions = "final",
    classProbs = TRUE,
    summaryFunction = twoClassSummary
  )

  # Entrenar modelo SVM
  svm_model <- train(class ~ ., data = filteredData, method = "svmRadial",
                     trControl = fitControl, preProcess = "scale", metric =
"ROC")
  models_list$svm_models[[i]] <- svm_model

  # Entrenar modelo Random Forest
  rf_model <- train(class ~ ., data = filteredData, method = "rf",
                    trControl = fitControl, metric = "ROC")
  models_list$rf_models[[i]] <- rf_model

  # Entrenar modelo J48
  j48_model <- train(class ~ ., data = filteredData, method = "J48",
                     trControl = fitControl, metric = "ROC")
  models_list$j48_models[[i]] <- j48_model

  # Evaluar el rendimiento del modelo en el conjunto de prueba para cada
modelo
  for(model_type in names(models_list)) {
    probabilities <- predict(models_list[[model_type]][[i]], testData, type
= "prob")
    predicted_classes <- predict(models_list[[model_type]][[i]], testData,
type = "raw")
    actual_classes <- testData$class
    cm <- confusionMatrix(data = predicted_classes, reference =
actual_classes)


    # Cálculo personalizado de MCC
    tp <- as.numeric(cm$table[1, 1])  # Verdaderos positivos
    tn <- as.numeric(cm$table[2, 2])  # Verdaderos negativos
    fp <- as.numeric(cm$table[1, 2])  # Falsos positivos
    fn <- as.numeric(cm$table[2, 1])  # Falsos negativos
    denom = sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
    if (denom == 0) {
      mcc = 0  # Prevenir división por cero
    } else {
      mcc = (tp * tn - fp * fn) / denom
    }
    roc <- roc(response = actual_classes, predictor = probabilities[,2])
    auc_value <- roc$auc

    accuracy <- (tp+tn)/(tp+tn+fp+fn)
    sensitivity <- cm$byClass['Sensitivity']
    specificity <- cm$byClass['Specificity']
    #recall <- tp/(tp+tn)
    precision <- cm$byClass['Precision']
    f1 <- cm$byClass['F1']
    geom_average = sqrt(sensitivity*specificity)

    performance_metrics[[model_type]][[i]] <- list(
      AUC = auc_value,
```

```
      Accuracy = accuracy,
      Sensitivity = sensitivity,
      Specificity = specificity,
      F1 = f1,
      MCC = mcc,
      #Recall = recall,
      Geom_average = geom_average,
      Precision = precision
    )
  }
}

print(performance_metrics)

# Inicializar una lista para almacenar las métricas promedio para cada
modelo
average_metrics <- list(svm = list(), rf = list(), j48 = list())

# Calcular el promedio de cada métrica para los modelos SVM
for(metric in c( "Accuracy","Precision",  "Sensitivity", "Specificity",
"Geom_average", "F1","AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$svm_models, function(x)
x[[metric]])
  average_metrics$svm[[metric]] <- mean(metric_values)
}

# Calcular el promedio de cada métrica para los modelos RF
for(metric in c( "Accuracy","Precision",  "Sensitivity", "Specificity",
"Geom_average", "F1","AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$rf_models, function(x)
x[[metric]])
  average_metrics$rf[[metric]] <- mean(metric_values)
}

# Calcular el promedio de cada métrica para los modelos J48
for(metric in c( "Accuracy", "Precision", "Sensitivity", "Specificity",
"Geom_average", "F1","AUC", "MCC")) {
  metric_values <- sapply(performance_metrics$j48_models, function(x)
x[[metric]])
  average_metrics$j48[[metric]] <- mean(metric_values)
}

# Imprimir las métricas promedio para cada modelo
print(average_metrics)

# Iniciar con un data frame vacío para almacenar los resultados
long_df_smote_ipf <- data.frame(Algorithm = character(),
                      Metric = character(),
                      Average_smote_ipf = numeric(),
                      stringsAsFactors = FALSE)

# Bucle a través de cada algoritmo
for(alg in names(average_metrics)) {
  # Para cada algoritmo, crear un data frame temporal de sus métricas
  temp_df <- data.frame(Algorithm = alg,
                        Metric = names(average_metrics[[alg]]),
                        Average_smote_ipf = unlist(average_metrics[[alg]]),
                        stringsAsFactors = FALSE)

  # Unir el data frame temporal al data frame largo
  long_df_smote_ipf <- rbind(long_df_smote_ipf, temp_df)
```

```
}

# Eliminar nombres de filas para asegurar que no se impriman
rownames(long_df_smote_ipf) <- NULL

# Imprimir el data frame limpio en formato largo
print(long_df_smote_ipf)
```