Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN CIENCIA E INGENIERÍA DE DATOS

Optimización Computacional de ReliefF: Estrategias basadas en Búsqueda Aproximada y Prototipado

Estudiante: Nicolás Aller Ponte

Dirección: Verónica Bolón Canedo

Samuel Suárez Marcote

A Coruña, julio de 2026.

"Me he pasado mi vida entera buscando el siguiente escalón."

A quienes me ayudaron a seguir hacia delante

Agradecimientos

A mi familia, pilar fundamental de todo lo que soy. Gracias en especial a papá y a mamá, quiero que os sintáis orgullosos de vuestro hijo, pero sobre todo de vosotros mismos, porque este logro tiene mucho más mérito vuestro que mío. Me habéis acompañado en cada paso, hemos crecido juntos como personas y quiero que sigamos juntos, siempre.

A Inés, que me empujó hacia adelante cuando ni yo mismo sabía si me quedaban fuerzas. Tú hiciste que cada momento fuera mejor y que nunca me faltaran amor y ganas de mirar al horizonte. Gracias.

A mis compañeros, que convirtieron la carrera en un camino precioso y, contra todo pronóstico, divertido, con vosotros hasta las doce horas en la biblioteca llegaban a tener su gracia. Gracias por empaparme de vuestro conocimiento y, sobre todo, de vuestro cariño.

A mis amigos, que siempre han estado y siempre estarán ahí para apoyarme, reírse conmigo y, cuando hizo falta, también para llorar.

Gracias, por último, a todo el profesorado, por brindarme un aprendizaje excepcional que valoraré siempre. Quiero hacer una mención especial a mis tutores, Verónica y Samuel, por la confianza y el apoyo incondicionales que me han demostrado. Os lo agradeceré eternamente.

Resumen

Ante el crecimiento exponencial del volumen de datos, la selección de características es indispensable, pero la complejidad cuadrática $\mathcal{O}(n^2 \cdot m)$ de los algoritmos basados en vecindarios, como ReliefF, resulta insostenible a gran escala, tanto en cómputo como en consumo energético, en contra de los principios de la IA Verde. Este trabajo propone dos variantes que reducen drásticamente ese coste sin alterar el criterio de relevancia, conservando la capacidad de ReliefF para detectar interacciones entre variables. HNSW-ReliefF sustituye la búsqueda exacta de vecinos por consultas de coste sublineal sobre un índice de grafo navegable jerárquico (HNSW), y Proto-ReliefF comprime el conjunto de entrenamiento en un número acotado de prototipos representativos mediante *clustering*. Sobre datos de hasta millones de instancias, donde ReliefF es inviable, ambas logran aceleraciones de más de dos órdenes de magnitud y reducen las emisiones en más de un 30%, manteniendo una calidad de selección estadísticamente equivalente a la del algoritmo original.

Abstract

Given the exponential growth in data volume, feature selection is indispensable, yet the quadratic complexity $\mathcal{O}(n^2 \cdot m)$ of neighbourhood-based algorithms such as ReliefF is unsustainable at scale, both in computation and energy consumption, against the principles of Green AI. This work proposes two variants that drastically reduce that cost without altering the relevance criterion, preserving ReliefF's ability to detect feature interactions. HNSW-ReliefF replaces the exact nearest-neighbour search with sublinear-cost queries over a Hierarchical Navigable Small World (HNSW) graph index, and Proto-ReliefF compresses the training set into a bounded number of representative prototypes through clustering. On datasets of up to millions of instances, where ReliefF is infeasible, both achieve speed-ups of more than two orders of magnitude and cut emissions by over one third, while keeping a selection quality statistically equivalent to that of the original algorithm.

Palabras clave:

- Selección de características
- ReliefF
- Búsqueda aproximada de vecinos
- HNSW
- IA Verde
- Prototipado
- Escalabilidad

Keywords:

- Feature Selection
- ReliefF
- Approximate Nearest Neighbors
- HNSW
- Green AI
- Prototyping
- Scalability

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

La Ciencia de Datos y la Inteligencia Artificial (IA) han vivido una transformación total en la última década. Ya no se sufre por la falta de datos, ahora el reto es procesar tal cantidad de información de forma eficiente. Cada día se afrontan conjuntos de datos que crecen de forma constante tanto en volumen como en dimensionalidad, aumentando la aparición del fenómeno conocido como "maldición de la dimensionalidad" [2]. Un ejemplo ilustrativo son las plataformas de vídeo o *e-commerce*, donde miles de productos y usuarios generan millones de interacciones diarias. Procesar tales magnitudes de información puede llegar a ser inasumible si no se hace una gestión precisa de qué variables tienen realmente relevancia. En estos entornos de dimensionalidad tan alta, muchos de los datos capturados podrían ser ruido o información redundante con otras variables y, dichos atributos, pueden llegar a entorpecer el aprendizaje de los modelos. Es por eso que, antes de aplicar cualquier arquitectura de aprendizaje automático, es crítico el preprocesado del dato.

Dentro del grupo de técnicas de preprocesamiento, la Selección de Características (*Feature Selection*) [3] es un mecanismo útil para reducir la complejidad del espacio de búsqueda. Las técnicas existentes se agrupan en tres familias según su relación con el modelo de aprendizaje [3]: los filtros, que valoran la relevancia de cada variable mediante criterios estadísticos independientes del clasificador; los métodos embebidos, que integran la selección dentro del propio entrenamiento del modelo; y los *wrappers*, que utilizan el rendimiento de un clasificador como criterio para evaluar subconjuntos de variables. De las tres familias, los filtros son los más adecuados cuando el volumen de datos es elevado, al no depender de un clasificador, evitan el coste de entrenar el modelo de forma repetida y ofrecen un preprocesado más rápido, escalable y reutilizable con cualquier modelo posterior. Por ese motivo, este trabajo se centra en la familia de filtros, entre los que destacan los métodos que se basan en instancias y, más específicamente, la familia Relief (entre cuyos métodos se encuentra ReliefF), ampliamente estudiada en la Ciencia de Datos. Esta familia de métodos está especialmente valorada por su capacidad de captar dependencias no lineales, aunque su adaptación a los actuales entornos

Big Data se ve frenada por un coste computacional que crece de forma cuadrática a medida que aumenta el número de muestras. En problemas reales, esto se traduce en tiempos de ejecución prohibitivos y en un consumo de recursos *hardware* difícilmente asumible. El procesamiento de tales cantidades de información genera una huella energética de gran calibre, que obliga a replantear el diseño de los algoritmos teniendo en cuenta el impacto ecológico de estos procesos.

Como se ha mencionado anteriormente, se debe considerar que este coste computacional no es solo temporal, sino que conlleva un impacto ambiental que a menudo se ignora. La preocupación por el impacto ecológico de la Inteligencia Artificial ha dado lugar a un movimiento emergente conocido como IA Verde [4], donde se considera el desarrollo de modelos y técnicas que mantengan un buen equilibrio entre la precisión de los métodos y la eficiencia energética. Diversos estudios recientes han cuantificado el impacto ambiental de entrenar modelos complejos. Strubell et al. [5] estimaron que el entrenamiento de grandes modelos de lenguaje puede generar hasta 284 toneladas de CO_2 equivalente, aproximadamente cinco veces las emisiones totales de un automóvil durante su vida útil, mientras que trabajos posteriores han documentado que modelos como GPT-3 produjeron más de 500 toneladas de CO_2 durante su entrenamiento [6]. Con estos datos se puede afirmar que optimizar la fase de selección de características no solo acelera el preprocesado, sino que contribuye directamente a la reducción del coste energético del procesamiento realizado por un modelo de Aprendizaje Automático.

Este trabajo se centra en solucionar esta problemática mediante el diseño, implementación y evaluación de dos estrategias de optimización del algoritmo ReliefF. La primera estrategia emplea estructuras de indexación basadas en búsqueda aproximada de vecinos más cercanos (*Approximate Nearest Neighbors*, ANN), específicamente mediante el algoritmo HNSW (*Hierarchical Navigable Small World*) [7], que permite reducir la complejidad de la búsqueda de vecinos de $\mathcal{O}(n^2)$ a $\mathcal{O}(n \log n)$ con un error reducido y controlado. La segunda estrategia se fundamenta en la selección de prototipos representativos del conjunto de datos [8], reduciendo así el espacio de búsqueda sin comprometer significativamente la calidad de la evaluación de características. Ambas aproximaciones son evaluadas a partir de tres bases: la calidad de la selección (mediante métricas de clasificación), la robustez ante perturbaciones (inyección controlada de ruido) y la escalabilidad computacional.

Conviene señalar que este Trabajo de Fin de Grado (TFG) se enmarca en la sección de desarrollo de algoritmos que sean eficientes energéticamente y ha contado con el apoyo de un contrato de la "Cátedra Inditex-UDC de IA en Algoritmos Verdes". Como muestra del compromiso con una investigación transparente y reproducible, se ha liberado el código fuente desarrollado como software libre y se encuentra disponible en el repositorio de GitHub[1].

[1] https://github.com/nicolasallerponte/ReliefF-Optimization

1.1 Objetivos del proyecto

Teniendo en cuenta el contexto descrito, se presentan a continuación los objetivos que estructuran este trabajo.

1.1.1 Objetivo principal

El objetivo principal de este trabajo es diseñar, implementar y evaluar versiones sostenibles de ReliefF que mantengan una calidad en la selección de características similar a la versión original del método, pero reduciendo de forma significativa tanto su coste computacional como el consumo energético asociado al proceso de selección de las características.

1.1.2 Objetivos específicos

Este objetivo general se desglosa en las siguientes metas de investigación, siendo también las distintas fases del trabajo realizado:

- Comprender en profundidad el funcionamiento de ReliefF y sus variantes, así como sus limitaciones en términos de complejidad computacional en escenarios de alta dimensionalidad y grandes volúmenes de datos.
- Diseñar e implementar una variante de ReliefF basada en estructuras de búsqueda aproximada de vecinos más cercanos, empleando índices del tipo HNSW con el objetivo de reducir el coste de la búsqueda de vecinos manteniendo una alta precisión.
- Diseñar e implementar una segunda variante de ReliefF basada en prototipos, en la que se sustituyan las instancias originales por un conjunto reducido de representantes de modo que se disminuya el tamaño efectivo del conjunto de datos sin una pérdida significativa de la información disponible.
- Estudiar la robustez de las variantes propuestas frente a distintos tipos de ruido, analizando cómo se ve afectada la capacidad tanto de ReliefF como de las variantes propuestas para identificar atributos relevantes en este tipo de entornos.
- Analizar la escalabilidad de las implementaciones desarrolladas en términos de tiempo de ejecución y uso de recursos, utilizando conjuntos de datos de tamaño creciente, comprobando su viabilidad en estos problemas de gran escala.

1.2 Planificación y costes del proyecto

El desarrollo de este TFG se estructuró en seis fases aproximadamente secuenciales, representadas en el diagrama de Gantt de la Figura 1.1.

1. **Revisión bibliográfica**: Estudio de ReliefF y sus variantes, algoritmos de búsqueda aproximada de vecinos y técnicas de selección de prototipos.
2. **Diseño de variantes**: Definición teórica y código de HNSW-ReliefF y Proto-ReliefF, con identificación de los hiperparámetros críticos de cada una.
3. **Implementación base**: Integración de ReliefF mediante *skrebate* y primera versión funcional de las variantes sobre conjuntos de datos de referencia.
4. **Optimización y evaluación**: *Grid search* de hiperparámetros, pruebas de robustez frente a ruido y análisis de escalabilidad.
5. **Validación estadística**: Comparación de las variantes frente a ReliefF mediante tests estadísticos.
6. **Análisis y documentación**: Interpretación de resultados, extracción de conclusiones y redacción de la memoria.

El diagrama de Gantt de la Figura 1.1 muestra la distribución temporal de las fases del proyecto. El trabajo se desarrolla a lo largo de un periodo de unos 10 meses aproximadamente, entre septiembre de 2025 y junio de 2026, con una carga de trabajo mayor en el tramo central, de diciembre a marzo, donde la implementación y experimentación se solapan parcialmente. El trabajo ha consistido en aproximadamente 290 días de trabajo con un total de 450 horas.

La Tabla 1.1 recoge una estimación de los costes de los recursos empleados y su coste asociado. El software utilizado es, en su totalidad, de código abierto, por lo que no genera coste de licencias. El coste del ingeniero se ha calculado tomando como referencia el salario mínimo fijado para un analista de datos en el XXI Convenio Colectivo Estatal de Empresas de Consultoría y Tecnologías de la Información (aproximadamente 24.105 € brutos anuales, equivalente a 13,40 €/hora), con una dedicación estimada de 450 horas. Para los dos directores se estima una dedicación de 45 horas cada uno, incluyendo reuniones de seguimiento y revisión del trabajo, a un coste de 20 €/hora.

Tipo de Recurso	Recurso	Coste (€)
Software	Python 3.11	0,00
Software	hnswlib	0,00
Software	scikit-learn	0,00
Software	Overleaf	0,00

Tabla 1.1 – *(viene de la página anterior)*

Tipo de recurso	Recurso	Coste (€)
Hardware	Ordenador Personal	900,00
Humanos	Ingeniero de Datos	6.030,00
Humanos	Director 1	900,00
Humanos	Director 2	900,00
Total		**8.730,00**

Tabla 1.1: Estimación de costes del proyecto

1.3 Tecnologías y herramientas

El desarrollo se ha realizado en Python 3.11 por su disponibilidad de librerías especializadas y bien mantenidas. Las principales librerías empleadas se describen a continuación; la memoria se ha redactado de forma independiente con Overleaf y LaTeX.

skrebate Implementación de ReliefF utilizada como base sobre la que trabajar y punto de comparación en los experimentos. Es compatible con scikit-learn, lo que facilita su evaluación.

scikit-learn 1.3+ Librería de Python que proporciona funcionalidades para preprocesado de datos y métricas de evaluación útiles para los experimentos (F1-Score, AUC). Se emplea también para la generación de *datasets* sintéticos.

hnswlib 0.8+ Construcción y consulta de índices HNSW sobre código C++, lo que permite tiempos de búsqueda efectivos incluso en alta dimensionalidad. Constituye el núcleo de la variante HNSW-ReliefF.

Numba 0.58+ Compilación JIT del núcleo de cómputo de las variantes a código máquina, eliminando la sobrecarga del intérprete de Python en el bucle de puntuación.

NumPy 1.24+ / Pandas 2.0+ Manipulación eficiente de matrices y gestión de los datasets empleados en los experimentos. También usado para los cálculos internos de los algoritmos.

Matplotlib 3.7+ Generación de todas las figuras relacionadas con la experimentación.

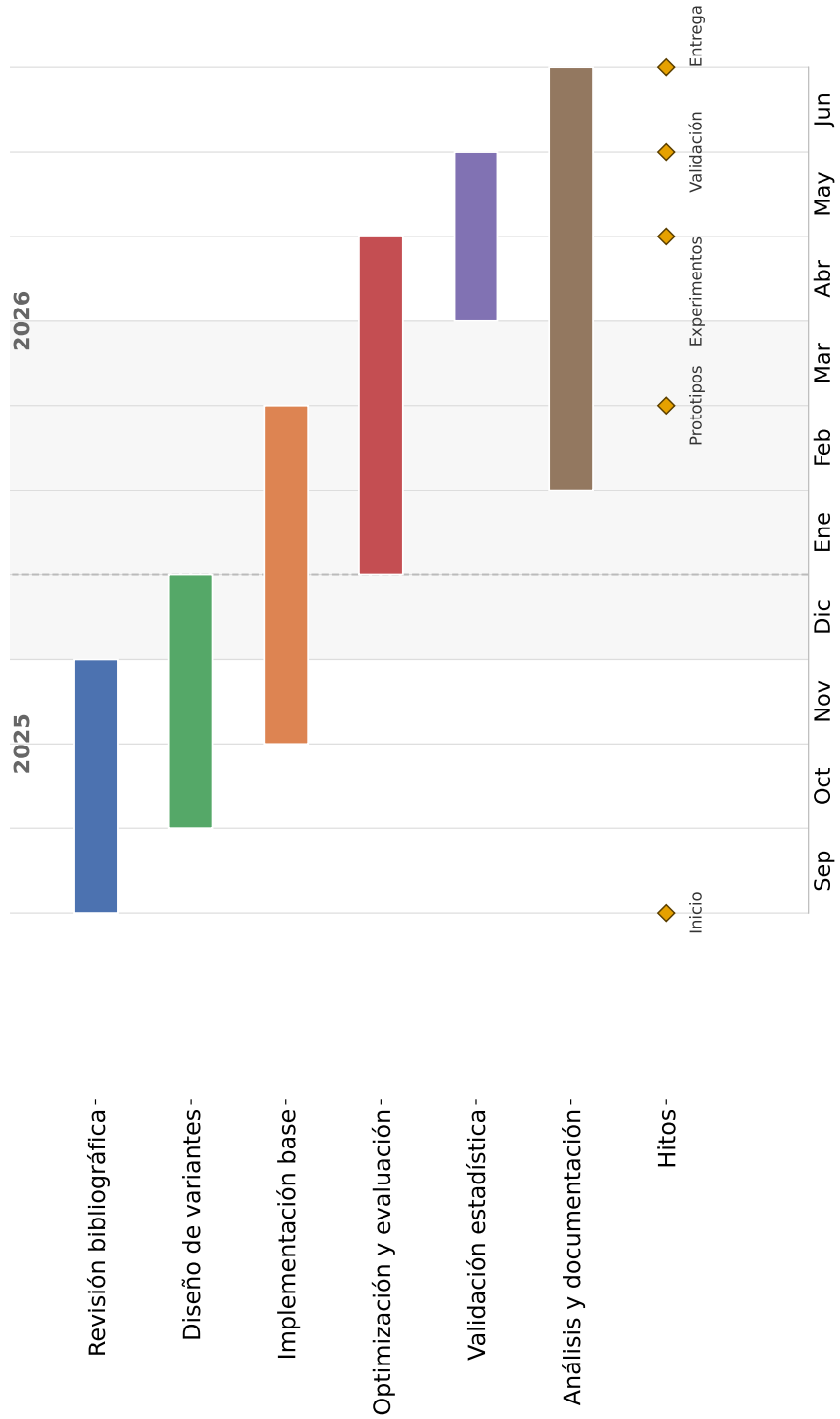


Figura 1.1: Diagrama de Gantt con la distribución temporal de las seis fases del proyecto y sus hitos principales, entre septiembre de 2025 y junio de 2026. El tramo central (diciembre a marzo) concentra la mayor carga, con la implementación y la experimentación solapadas.

El código completo se distribuye como software libre bajo licencia MIT[2].

1.4 Estructura de la memoria

A continuación se presenta un breve resumen de los cinco capítulos principales, estructurados de la siguiente manera en este TFG:

- **Capítulo 1: Introducción.** Se sitúa el trabajo en su contexto, se expone la problemática que lo motiva y se definen los objetivos planteados. Se describen también las herramientas y tecnologías empleadas durante el desarrollo y la planificación que ha guiado el proyecto.
- **Capítulo 2: Marco teórico y estado del arte.** Se recorren los fundamentos teóricos sobre los que se asienta el trabajo. Se estudia la selección de características basada en vecinos y la evolución de ReliefF desde sus orígenes hasta sus variantes modernas. Se analizan también los grafos HNSW como estructura de indexación para búsqueda aproximada, y se revisan los métodos de reducción de conjuntos de entrenamiento mediante prototipos que inspiran la segunda propuesta del trabajo.
- **Capítulo 3: Metodología.** Se presenta el núcleo de la contribución del trabajo. Se describe el diseño de HNSW-ReliefF, que integra búsqueda aproximada de vecinos en el esquema de puntuación de ReliefF, y de Proto-ReliefF, que reduce el conjunto de referencia a un subconjunto de prototipos antes de aplicar el algoritmo. Para cada variante se justifican las decisiones tomadas y se caracteriza su espacio de hiperparámetros.
- **Capítulo 4: Experimentación y resultados.** Este es el núcleo práctico de la investigación. Se describe la experimentación, los conjuntos de datos empleados y la configuración del ajuste de hiperparámetros realizado. A continuación se presentan y discuten los resultados obtenidos.
- **Capítulo 5: Conclusiones.** Cierra la memoria recogiendo las principales ideas extraídas del trabajo experimental. Se valora el cumplimiento de los objetivos planteados, se identifican las limitaciones del enfoque y se proponen las posibles líneas sobre las que realizar el trabajo futuro.

[2] https://github.com/nicolasallerponte/ReliefF-Optimization/blob/main/LICENSE

Capítulo 2

Marco Teórico y Estado del Arte

La selección de características es uno de los problemas centrales del aprendizaje automático y ha generado una extensa investigación desde los años noventa [9]. La relevancia de este problema ha crecido de forma proporcional al volumen de los conjuntos de datos disponibles, a medida que los entornos de alta dimensionalidad y gran escala se han convertido en la norma. La capacidad para identificar qué variables aportan información útil ha pasado de ser una técnica auxiliar en el desarrollo a convertirse en un componente crítico del diseño de sistemas de aprendizaje. En este capítulo se revisan los fundamentos y el estado actual de los tres pilares sobre los que se apoya este trabajo, los cuales son la selección de características basada en vecinos y la familia Relief, los métodos de búsqueda aproximada de vecinos y las técnicas de reducción del conjunto de entrenamiento mediante prototipos.

2.1 Selección de Características

Dado un conjunto de datos con m variables, el objetivo de la selección de características es encontrar el subconjunto $S \subseteq \{1, \ldots, m\}$ de tamaño $d < m$ que preserve, en la medida de lo posible, la información relevante para una tarea concreta de aprendizaje automático. El número de subconjuntos posibles crece exponencialmente con m, lo que hace inviable la búsqueda exhaustiva en espacios de alta dimensionalidad [9]. Cuando el número de variables crece respecto al número de instancias disponibles, los modelos tienden a memorizar el conjunto de entrenamiento en lugar de aprender patrones generalizables, fenómeno conocido como *overfitting*, al que se suma la maldición de la dimensionalidad [2], que degrada la capacidad discriminativa de los métodos basados en distancias al homogeneizar las diferencias entre puntos cercanos y lejanos. Estudios han demostrado que aplicar selección de características previa al entrenamiento puede mejorar la precisión de clasificación hasta en un 15,55 % respecto a entrenar con el conjunto completo de variables [10]. Ese mismo trabajo advierte, no obstante, que la selección no debe ser demasiado agresiva, ya que eliminar variables que sí

aportaban información hace que la precisión se desplome drásticamente.

Los métodos de selección de características se agrupan en tres familias según la relación que establecen con el clasificador [9]. Los métodos de *filtro* evalúan la relevancia de cada variable con independencia del algoritmo de inducción, empleando criterios estadísticos que los hacen altamente escalables y aplicables a grandes conjuntos de datos. Los métodos *wrapper* incorporan el propio clasificador como función de evaluación, probando subconjuntos de variables y midiendo directamente la precisión del modelo, lo que los hace más precisos pero computacionalmente costosos al requerir entrenar el clasificador en cada iteración. Los métodos *embebidos* integran la selección dentro del proceso de entrenamiento, como ocurre en los árboles de decisión o en la regularización L1 (Lasso), donde la penalización sobre los coeficientes actúa directamente como mecanismo de selección. En la práctica, los métodos de filtro son los más utilizados en escenarios de alta dimensionalidad por su independencia respecto al algoritmo de inducción y su eficiencia computacional.

2.1.1 Relevancia de las características

Para poder seleccionar variables de manera correcta y rigurosa es necesario definir qué significa que una variable sea relevante. Kohavi y John [11] formalizaron este concepto distinguiendo tres categorías: una variable es *fuertemente relevante* si su eliminación del conjunto completo produce una degradación del rendimiento del clasificador, porque contiene información que ninguna otra variable puede aportar; es *débilmente relevante* si no pertenece a la primera categoría pero existe algún subconjunto en el que su inclusión mejora el rendimiento, siendo su información redundante con otras variables pero no prescindible en todos los contextos; y es *irrelevante* si su presencia o ausencia no afecta al rendimiento en ningún subconjunto posible.

Esta distinción explica el límite de los criterios univariados, los más extendidos en la práctica. La información mutua $I(f; y)$, que mide cuánto reduce la incertidumbre sobre la clase y el conocer el valor de una variable f, es un ejemplo representativo. Al evaluar cada variable de forma aislada, identifica bien las fuertemente relevantes, pero tiende a descartar las débilmente relevantes cuya utilidad es condicional a otras y a retener variables irrelevantes correlacionadas con la clase de forma espuria. Esta limitación es precisamente la que motiva los métodos basados en vecindario como ReliefF.

2.1.2 Interacción entre características

La limitación más importante de los métodos univariados no es de precisión, sino de cómo actúan, ya que son incapaces de detectar variables cuya relevancia es condicional a otras [12]. Una variable f_a puede ser individualmente irrelevante, en el sentido de que su información

mutua con la clase sea prácticamente nula $I(f_a; y) \approx 0$, y resultar imprescindible si su combinación con f_b determina completamente la clase. Este fenómeno se conoce como interacción entre variables y es uno de los problemas más estudiados en la selección de características.

El ejemplo canónico es el problema XOR, donde $y = f_a \oplus f_b$. Si se mira solo la columna f_a, los ceros y unos se reparten a partes iguales entre las dos clases, ocurriendo lo mismo con f_b. Ninguna de las dos variables vistas por separado permite distinguir una clase de la otra, como muestra la Tabla 2.1. Sin embargo, conociendo ambas a la vez, la clase queda completamente determinada: cuando f_a y f_b coinciden, $y = 0$; cuando difieren, $y = 1$.

f_a	f_b	$y = f_a \oplus f_b$
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 2.1: Tabla de verdad del problema XOR.

El conjunto de datos CorrAL [13], en el que la clase viene determinada por la expresión lógica $(f_1 \wedge f_2) \vee (f_3 \wedge f_4)$, es un caso de uso clásico para evaluar la capacidad de detección de estas interacciones. Ninguna de las cuatro variables es individualmente discriminativa, siendo relevantes únicamente en combinación con su pareja. Cualquier filtro univariado descartaría las cuatro, eliminando precisamente las variables relevantes del problema. Este conjunto se emplea en la experimentación de este TFG precisamente por esta propiedad.

2.2 La familia Relief

La familia Relief constituye uno de los enfoques más estudiados dentro de los filtros de selección de características [14]. A diferencia de los métodos univariados, evalúa la relevancia de cada variable en función del comportamiento local de los datos, lo que le permite capturar las interacciones entre variables descritas en la sección anterior.

2.2.1 Relief y ReliefF

El origen de la familia se remonta a 1992, cuando Kira y Rendell propusieron Relief [15]: un algoritmo que, para cada instancia x_i del conjunto $X = \{x_1, \ldots, x_n\}$ (el vector de las m variables de la muestra i), toma su vecino más cercano de la misma clase (*nearest hit*, H) y su vecino más cercano de clase distinta (*nearest miss*, M), actualizando el peso de cada variable según la siguiente ecuación:

$$W(f) \mathrel{-}= \text{diff}(f, x_i, H) - \text{diff}(f, x_i, M) \tag{2.1}$$

donde $\text{diff}(f, x_i, x_j)$ es la diferencia normalizada entre los valores de la variable f en ambas instancias. Si dos muestras de clases distintas son vecinas pero una variable les asigna valores similares, esa variable no contribuye a distinguirlas; a la inversa, si dos muestras de la misma clase son vecinas pero la variable las separa, estaría introduciendo variación intraclase donde no debería haberla. Tras iterar sobre todas las muestras, las variables con mayor peso acumulado son las más discriminativas. El algoritmo presenta dos limitaciones claras, por un lado, está restringido a problemas binarios y, por otro, resulta sensible al ruido al basarse en un único par de vecinos.

Kononenko propuso en 1994 la extensión ReliefF [16], que aborda ambas limitaciones de forma simultánea. La generalización multiclase sustituye el único *nearest miss* por los k vecinos más cercanos de cada clase distinta a x_i, ponderando su contribución según la probabilidad a priori de cada clase para que las clases más frecuentes no dominen la actualización de pesos. También sustituye el único par de vecinos por los k vecinos más cercanos tanto de la clase propia como de cada clase ajena, promediando sus contribuciones para obtener una estimación más robusta frente al ruido. Formalmente:

$$W(f) \mathrel{+}= -\frac{1}{n \cdot k} \sum_{j \in \text{Hits}(i)} \text{diff}(f, x_i, x_j) + \sum_{c \neq y_i} \frac{P(c)}{1 - P(y_i)} \cdot \frac{1}{n \cdot k} \sum_{j \in \text{Misses}_c(i)} \text{diff}(f, x_i, x_j) \tag{2.2}$$

donde n es el número de instancias, $P(c)$ es la probabilidad estimada de la clase c e y_i es la clase de x_i. El factor $\frac{P(c)}{1-P(y_i)}$ normaliza la contribución de cada clase ajena para que el total de penalizaciones sea comparable al del algoritmo binario original, independientemente del número de clases. Una consecuencia importante de esta formulación es que ReliefF puede asignar pesos altos a variables cuya relevancia es condicional a otras. Por ejemplo, si f_a solo discrimina cuando f_b toma un valor concreto, esa interacción se refleja de forma consistente en el término de los *misses* a lo largo de las instancias donde la interacción es activa. Al promediar sobre k vecinos, ReliefF estabiliza la estimación y destaca esas interacciones locales de forma robusta, como demuestra el análisis teórico y empírico de Robnik-Sikonja y Kononenko [17]. El Algoritmo 1 resume el procedimiento completo.

Algoritmo 1: ReliefF

Input: Conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^{n}$, número de vecinos k

Output: Vector de pesos $W \in \mathbb{R}^m$

$W(f) \leftarrow 0$ **para** $f = 1, \ldots, m$

for $i \leftarrow 1$ **to** n **do**

  Buscar los k *nearest hits* H_j (misma clase que x_i)

  foreach *clase* $c \neq y_i$ **do**

    Buscar los k *nearest misses* $M_j(c)$ de la clase c

  for $f \leftarrow 1$ **to** m **do**

$$W(f) \mathrel{-}= \frac{1}{n \cdot k} \sum_{j=1}^{k} \text{diff}(f, x_i, H_j)$$

$$W(f) \mathrel{+}= \sum_{c \neq y_i} \frac{P(c)}{1 - P(y_i)} \cdot \frac{1}{n \cdot k} \sum_{j=1}^{k} \text{diff}(f, x_i, M_j(c))$$

return W

2.2.2 Variantes de la familia

La influencia de ReliefF ha generado durante más de dos décadas una línea de trabajo constante. Las principales variantes comparten el principio de evaluación basado en *nearest hit* y *nearest miss* pero se diferencian en la definición del vecindario o en el tipo de tarea que abordan [18].

- **RReliefF** [17]: adapta el mecanismo al caso de regresión, ponderando la contribución de los vecinos en función de la diferencia entre valores de la variable objetivo continua en lugar de por pertenencia a una clase.
- **SURF** [19]: elimina el parámetro k y define el vecindario a partir de un umbral de distancia global calculado como la media de distancias entre instancias, eliminando así la necesidad de elegir el número de vecinos.
- **SURF*** [20]: extiende SURF incorporando las instancias más distantes como información adicional para detectar variables que separan regiones alejadas del espacio de características.
- **MultiSURF** [21]: en lugar de un umbral global, utiliza un umbral adaptativo dependiente de cada instancia. Esto permite que el vecindario representativo dependa de la densidad local del entorno de cada muestra.

- **TuRF** [22]: aplica ReliefF en rondas sucesivas, eliminando en cada iteración las variables de menor peso y recalculando sobre el subconjunto restante, mejorando la detección en problemas con alta proporción de ruido.

Las variantes basadas en umbral como SURF, SURF* y MultiSURF, eliminan la dependencia del parámetro k al definir el vecindario mediante una distancia de referencia, pero no reducen la complejidad algorítmica. Esto se debe a que para determinar qué instancias se sitúan dentro del umbral es igualmente necesario calcular las distancias desde cada punto al resto del conjunto. TuRF mejora la detección en escenarios ruidosos a costa de multiplicar el coste por el número de rondas de eliminación. Mientras, RReliefF extiende el algoritmo a tareas de regresión manteniendo la misma estructura y complejidad. En conjunto, las variantes de la familia han abordado la mejora de la calidad de la selección, la robustez al ruido y la generalización a nuevas tareas, pero ninguna ha sustituido ni aligerado el mecanismo de búsqueda de vecinos. La revisión de Urbanowicz et al. [18] confirma que esta limitación es el principal factor que restringe la aplicabilidad de la familia Relief en entornos de gran escala, identificando la eficiencia computacional como dirección de trabajo abierta.

2.2.3 Coste computacional de la búsqueda de vecinos

Tanto en su forma original como en sus variantes, el paso central de ReliefF es la búsqueda de los k vecinos más cercanos de cada instancia separados por clase. Para cada muestra x_i del conjunto de entrenamiento, el algoritmo necesita encontrar las k instancias más próximas de su misma clase y las k instancias más próximas de cada clase distinta. Con búsqueda exacta, esto implica comparar x_i contra todas las demás instancias del conjunto, calculando $\frac{n(n-1)}{2}$ distancias en total, lo que supone una complejidad $\mathcal{O}(n^2 \cdot m)$, donde m es el número de variables. Como consecuencia directa, al duplicar el número de muestras el tiempo de cómputo se cuadruplica.

Este efecto se agrava con la dimensionalidad por dos razones. En primer lugar, el coste de calcular cada distancia crece linealmente con m. En segundo lugar, en espacios de alta dimensión las distancias tienden a homogeneizarse, la diferencia relativa entre el vecino más cercano y el más lejano se reduce progresivamente, fenómeno conocido como concentración de la medida. Esto exige métricas y estrategias de búsqueda adaptadas, ya que el vecindario se vuelve menos informativo y más costoso de calcular a la vez. El coste temporal es, por tanto, solo una de las caras del problema, ya que la alta dimensionalidad degrada a la vez la utilidad del vecindario y el coste de calcularlo.

A modo de referencia, para $n = 10^4$ muestras y $m = 100$ variables el número de operaciones de distancia supera los 5×10^9; para $n = 10^5$ ya alcanza los 5×10^{11}. La Figura 2.1 ilustra este crecimiento con datos obtenidos durante la experimentación de este trabajo. Para

conjuntos a gran escala, ReliefF exacto requiere un tiempo de cómputo inabordable en *hardware* convencional; además, una barrera de memoria hace inviable la ejecución antes incluso de alcanzar esos tamaños.

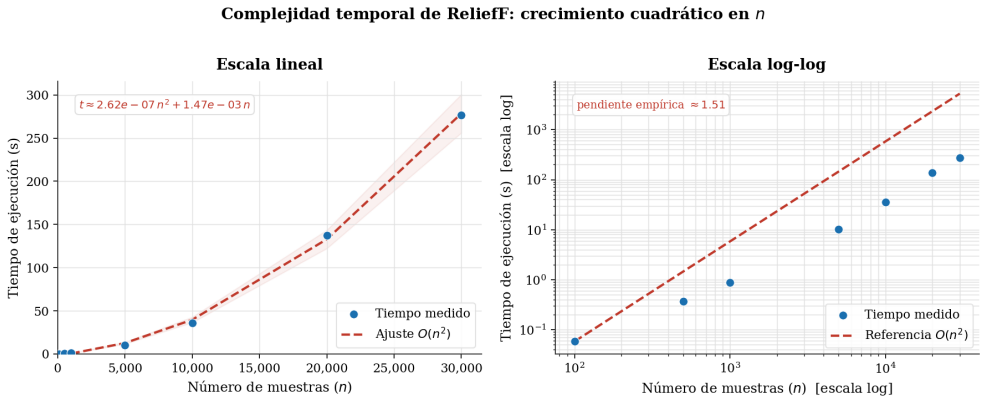


Figura 2.1: Tiempo de ejecución de ReliefF en función del número de muestras ($m = 20$ variables, $k = 10$ vecinos), evidenciando el crecimiento cuadrático de la búsqueda exacta de vecinos.

Más allá del tiempo, la búsqueda exacta plantea también un problema de memoria. Para garantizar que los k vecinos recuperados son realmente los más cercanos, es necesario materializar la matriz de distancias en memoria, cuyo tamaño crece con complejidad $\mathcal{O}(n^2)$. Para $n = 10^5$ muestras, esto supone almacenar del orden de 10^{10} valores, lo que excede la RAM disponible en la mayoría del *hardware* convencional e impide completar el proceso de selección.

2.2.4 Trabajos previos en escalabilidad de ReliefF

El coste cuadrático de ReliefF ha motivado varias líneas de trabajo orientadas a reducirlo sin alterar el criterio de actualización de pesos. Huang et al. [23] identifican la búsqueda de vecinos como cuello de botella principal y proponen una implementación optimizada que reutiliza cómputos de distancias, obteniendo mejoras de rendimiento significativas en conjuntos de $n \approx 10^4$ muestras sin modificar el resultado final. Esta aproximación reduce los factores constantes del algoritmo exacto pero no cambia su complejidad asintótica, además de mantener un uso de memoria RAM desorbitado en problemas de alta escala.

DiReliefF [24] reformula ReliefF para Apache Spark, expresando la búsqueda de vecinos y la actualización de pesos como operaciones de agregación sobre RDDs distribuidos. Esto permite procesar conjuntos del tamaño de ECBDL14 (33,6 millones de instancias, 632 variables), imposibles de manejar con ReliefF clásico en una sola máquina, con escalabilidad lineal en n y en el número de variables sobre el clúster. Sin embargo, el coste cuadrático sigue presente

en cada nodo, solo que se reparte entre máquinas, requiriendo infraestructura distribuida con un número de nodos proporcional a la carga.

Una línea diferente usa la propia lógica de vecindario de ReliefF para comprimir el conjunto de instancias antes de la selección. Abbasi y Rahmani [25] calculan pesos de instancias análogos a los pesos de variables de ReliefF y retienen únicamente las instancias de mayor peso, comprimiendo el conjunto a un porcentaje configurable. La extensión IFSB-ReliefF [26] integra selección de instancias y de variables en un único proceso por lotes paralelizable, mostrando reducciones de tiempo sustanciales en conjuntos de gran escala. Estos trabajos son los más cercanos en motivación a las propuestas de este TFG, aunque difieren en el mecanismo de reducción ya que utilizan el criterio de ReliefF para guiar qué instancias descartar, mientras que Proto-ReliefF aplica *clustering* externo para preservar la estructura de distribución de clases de forma independiente al criterio de relevancia.

El único trabajo que incorpora una estructura de índice espacial en la búsqueda de vecinos de ReliefF es el algoritmo de k-d tree aleatorizado propuesto por Xu et al. [27], que sustituye la búsqueda lineal por un árbol de partición espacial para datos de alta dimensionalidad. Como se detalla en la Sección 2.3, los árboles de partición espacial degeneran a búsqueda exhaustiva a partir de aproximadamente 20 dimensiones, lo que limita su eficacia en los espacios de características habitualmente densos de la selección de variables. Ningún trabajo previo ha integrado estructuras de índice de la generación actual, como grafos de proximidad jerárquicos (HNSW), *hashing* sensible a la localidad (LSH) o índices de archivo invertido (IVF), en la búsqueda de vecinos de ReliefF o de alguna de sus variantes.

Para abordar las limitaciones que persisten en la literatura se plantean dos estrategias complementarias. La primera consiste en renunciar a la exactitud de la búsqueda y emplear vecinos aproximados cuya recuperación tenga un coste mucho menor, este es el enfoque de ANN. La segunda consiste en reducir el número de instancias sobre las que se realiza esa búsqueda, comprimiendo el conjunto de datos en un conjunto de representantes o prototipos. Ambas aproximaciones atacan las mismas limitaciones de ReliefF pero desde ángulos distintos, constituyendo las dos propuestas desarrolladas en este trabajo.

2.3 Búsqueda Aproximada de Vecinos

ANN parte de la premisa de que en la mayoría de aplicaciones no es necesario encontrar el vecino exactamente más cercano, sino uno suficientemente cercano. Aceptar una pequeña tolerancia en la precisión permite diseñar estructuras de índices que evitan comparar cada instancia contra todas las demás, reduciendo la complejidad de $\mathcal{O}(n^2)$ a valores considerablemente menores en la práctica [28]. Hoy en día se distinguen tres familias principales de métodos ANN, cada una con ventajas y limitaciones diferentes.

Los *árboles de partición espacial*, como k-d trees y ball trees, dividen el espacio de características recursivamente en regiones mediante hiperplanos y podan ramas enteras durante la búsqueda cuando se puede garantizar que no contienen vecinos más cercanos que el mejor candidato conocido. Esta poda es muy efectiva en baja dimensionalidad, pero pierde toda su eficacia cuando el número de variables crece, ya que en espacios de alta dimensión la distancia al hiperplano separador resulta menor que la distancia al vecino más cercano, lo que impide podar cualquier rama. A partir de unas 20 dimensiones, estos métodos degeneran prácticamente a búsqueda exhaustiva [29].

Los métodos de *hashing* sensible a la localidad (LSH, *Locality-Sensitive Hashing*) proyectan los puntos en un espacio de menor dimensión mediante familias de funciones *hash* diseñadas para que puntos cercanos en el espacio original colisionen en el mismo cubo con alta probabilidad. Esto permite realizar búsquedas sublineales sin construir ninguna estructura de árbol. Su principal limitación práctica es que requieren un ajuste cuidadoso del número de tablas *hash* y del tamaño de las proyecciones. Esto se debe a que un número insuficiente de tablas genera fallos de colisión que degradan el *recall*, mientras que un número excesivo dispara el coste de memoria y consulta.

Los *grafos de proximidad* construyen grafos donde cada nodo está conectado a sus vecinos más cercanos y la búsqueda navega a través de ese grafo desde un punto de entrada hasta converger en la vecindad del punto buscado. Esta familia ha desplazado a las anteriores como referencia en *benchmarks* de ANN gracias a su mejor equilibrio entre precisión y velocidad en escenarios de alta dimensionalidad [28]. Dentro de ella, el algoritmo HNSW [7] representa el estado del arte.

2.3.1 Justificación de la elección de HNSW

HNSW organiza los nodos en una jerarquía de capas. Durante la construcción, cada nodo se inserta en la capa base y, con probabilidad decreciente, en capas superiores. Las capas altas contienen pocos nodos con conexiones de largo alcance que permiten atravesar el grafo rápidamente, mientras que la capa base contiene todos los nodos con conexiones locales de alta precisión. Cuando se inserta un nuevo nodo, el algoritmo navega desde la capa más alta hasta la base, seleccionando en cada capa los M vecinos más cercanos con los que establecerá conexiones; M, el parámetro de conectividad del grafo, fija cuántas aristas mantiene cada nodo y, con ello, el equilibrio entre densidad del grafo y consumo de memoria. El parámetro *ef_construction* controla el tamaño de la lista de candidatos durante la inserción donde un valor mayor produce conexiones de mejor calidad a costa de mayor tiempo de construcción. En conjunto, el coste de construcción del índice es $\mathcal{O}(n \log n)$.

Durante la consulta, la búsqueda parte del nodo de entrada en la capa más alta y desciende progresivamente, usando cada capa para aproximarse a la región del espacio donde se encuen-

tra el punto de consulta antes de refinar la búsqueda en la capa inferior, como se muestra de forma visual en la Figura 2.2. El parámetro *ef_search* controla el tamaño de la lista de candidatos durante la consulta y puede ajustarse sin necesidad de reconstruir el índice, permitiendo modificar el balance precisión-velocidad en tiempo de ejecución. El coste de cada consulta es $\mathcal{O}(\log n)$, transformando el cuello de botella cuadrático de ReliefF en un proceso logarítmico.

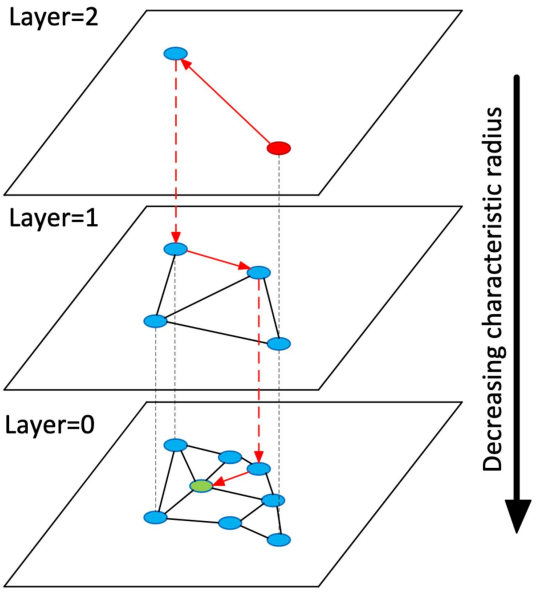


Figura 2.2: Estructura jerárquica de HNSW. La búsqueda parte del nodo de entrada en la capa superior (rojo) y desciende progresivamente hacia capas con conexiones más locales hasta localizar el vecino más cercano (verde) en la capa base. Fuente: [1].

HNSW es también eficiente en memoria, con un consumo de $\mathcal{O}(n \cdot M)$ que crece linealmente frente al $\mathcal{O}(n^2)$ de la búsqueda exacta y soporta actualizaciones incrementales del índice. Trabajos recientes han explorado variantes orientadas a la optimización automática de sus parámetros [30], consolidando a HNSW como la estructura de índice de referencia para búsqueda aproximada de vecinos en el contexto del aprendizaje automático.

La elección de HNSW sobre otras estructuras ANN disponibles se fundamenta en las limitaciones operativas de las principales alternativas. Los métodos basados en LSH, por su ya mencionada sensibilidad al número de tablas *hash* y a los parámetros de proyección, exigen un ajuste fino que HNSW no necesita. FAISS [31], la biblioteca de referencia de Meta para búsqueda a escala masiva, combina índices de archivo invertido con cuantización del producto para lograr una eficiencia de memoria sobresaliente en conjuntos de cientos de millones de vectores; sin embargo, la cuantización introduce una degradación de precisión que se vuelve apreciable en espacios de dimensionalidad baja o media, habituales en selección de características. ScaNN [32], de Google, mejora ese compromiso entre *recall* y velocidad mediante una

cuantización más elaborada, pero exige una fase de entrenamiento previa sobre el conjunto de referencia que penaliza los escenarios con datos cambiantes o de tamaño moderado.

Los *benchmarks* de ANN-Benchmarks [33], que evalúan más de una docena de implementaciones sobre conjuntos con distintas dimensionalidades, tamaños y métricas de distancia, muestran de forma consistente que HNSW alcanza los mejores resultados al ponderar *recall* y velocidad de consulta en la mayoría de escenarios. Análisis comparativos recientes entre métodos de grafo y de partición [34] confirman que los grafos de proximidad superan a los métodos basados en cuantización o partición cuando el objetivo es maximizar el *recall* sin preprocesamiento adicional. Estudios sobre el comportamiento de HNSW en función de la dimensionalidad intrínseca [35] indican además que el *recall* se mantiene estable en los espacios de características de dimensionalidad moderada habituales en selección de variables, a diferencia de LSH, cuyo *recall* decrece con la dimensionalidad si el número de tablas no se incrementa de forma proporcional. Adicionalmente, HNSW presenta ventajas para el contexto de este trabajo ya que no requiere ninguna fase de entrenamiento previo, admite inserción de nuevos nodos sin reconstrucción completa del índice y el balance precisión-velocidad se ajusta en tiempo de ejecución modificando *ef_search* sin necesidad de reconstruir el grafo.

2.4 Métodos basados en prototipos

Los métodos basados en prototipos abordan el problema de la escalabilidad desde una perspectiva diferente a la de ANN. En lugar de acelerar la búsqueda de vecinos, reducen el número de instancias sobre las que esa búsqueda se realiza. La idea central es comprimir el conjunto de entrenamiento en un conjunto de representantes que resumen la estructura del espacio original con muchas menos instancias. Si el conjunto original tiene n muestras y se reduce a $p \ll n$ prototipos, el coste de la búsqueda exacta pasa de $\mathcal{O}(n^2)$ a $\mathcal{O}(p^2)$, con una reducción del coste computacional que puede ser de varios órdenes de magnitud.

La forma más habitual de obtener prototipos es mediante algoritmos de *clustering*. El algoritmo *K-Means* [36] agrupa las instancias en p *clusters* y utiliza el centroide de cada uno como prototipo, capturando la distribución global del espacio con una fracción de las instancias originales. Una variante más adecuada para conjuntos de datos grandes es *MiniBatchK-Means* [37], que aproxima *K-Means* procesando los datos en minilotes en lugar de sobre el conjunto completo en cada iteración, reduciendo el coste de esta fase de compresión sin degradar apreciablemente la calidad de los centroides obtenidos.

Sin embargo, los centroides de *K-Means* son medias aritméticas de los puntos de cada *cluster* y, en la mayoría de los casos, no corresponden a ninguna instancia real del conjunto de datos original. Esto puede introducir puntos artificiales en regiones con distribuciones irregulares o con variables discretas. En un conjunto con variables binarias o categóricas, el

centroide puede producir valores decimales que no corresponden a ninguna categoría válida, distorsionando las distancias calculadas durante la búsqueda de vecinos. Una solución a este problema es aplicar un paso de *snap-to-grid*, sustituyendo cada centroide abstracto por la instancia real más cercana del conjunto original, garantizando, de esta forma, que los prototipos sean puntos válidos del espacio de características.

K-Medoids [38] aborda este problema en su construcción, restringiendo los prototipos a instancias reales del conjunto de entrenamiento. En lugar de calcular centroides, elige como representante de cada *cluster* la instancia que minimiza la distancia media al resto de miembros de ese *cluster*. Esto garantiza que los prototipos respetan automáticamente cualquier restricción del espacio de características, siendo especialmente relevante cuando las variables tienen restricciones de dominio o la métrica de distancia no es euclídea. El coste de optimización de *K-Medoids* es mayor que el de *K-Means*, aunque existen implementaciones eficientes que lo hacen viable para diferentes tamaños de problema.

Una limitación compartida entre ambos enfoques de *clustering* es que la compresión puede eliminar instancias que delimitan fronteras de decisión relevantes para la selección de características, especialmente en regiones poco densas del espacio. Para mitigar este efecto, se puede incorporar un paso de refinamiento mediante *Learning Vector Quantization* (LVQ) [39], el cual es una técnica de aprendizaje supervisado que ajusta la posición de los prototipos de forma iterativa, acercándolos a las instancias de su propia clase y alejándolos de las de clases distintas. Este refinamiento busca que los prototipos no solo representen la distribución estadística de los datos sino que queden posicionados en las zonas del espacio donde la discriminación entre clases es más informativa para la estimación de relevancia de ReliefF.

Comprimir el conjunto de instancias mediante agrupamiento y estimar después la relevancia sobre la representación reducida es una estrategia con precedentes recientes, ajenos a la familia Relief. *LSH-IS* [40] aplica *hashing* sensible a la localidad para seleccionar instancias representativas y emplea la correlación de Pearson como filtro sobre el conjunto de muestras reducido, demostrando que un criterio de selección de características puede operar sobre datos comprimidos sin pérdida significativa de calidad. Los métodos basados en bolas granulares [41, 42] construyen prototipos mediante agrupamiento difuso y definen criterios de información mutua directamente sobre esa representación, realizando la selección sobre los grupos en lugar de sobre las instancias individuales. En ambos casos la reducción es un paso previo independiente del criterio de selección, lo que valida estructuralmente el enfoque de compresión seguida de estimación de relevancia.

Capítulo 3

Metodología

En el capítulo anterior se identificó la búsqueda exacta de vecinos como el cuello de botella central de ReliefF. Su coste $\mathcal{O}(n^2 \cdot m)$ de comparar cada instancia contra todas las demás hace inviable su aplicación directa a conjuntos de datos de escala moderada o grande. En este capítulo se describen las dos propuestas algorítmicas desarrolladas para abordar esa limitación, detallando su diseño, parámetros y las decisiones de implementación que determinan su comportamiento.

3.1 Motivación y enfoque

Ambas propuestas parten de la misma premisa y es que el criterio de actualización de pesos de ReliefF, definido en la Ecuación (2.2), es sólido y no necesita modificación. Cualquier simplificación del mecanismo de estimación comprometería la capacidad de detectar interacciones entre variables, que es precisamente la propiedad que diferencia a ReliefF de los filtros univariados. El objetivo, por tanto, no es rediseñar el algoritmo sino identificar qué parte del proceso es computacionalmente dominante y sustituirla por una alternativa más eficiente.

Esa parte es la búsqueda de vecinos. En cada iteración del bucle principal, ReliefF localiza los k vecinos más cercanos de cada instancia x_i separados por clase, comparando x_i con todas las demás instancias del conjunto. El proceso se repite n veces y, en su forma exacta, no admite ninguna optimización sin renunciar a alguna garantía. Cada una de las dos propuestas de este trabajo renuncia a una garantía diferente para ganar eficiencia.

HNSW-ReliefF renuncia a la exactitud de la búsqueda. En lugar de encontrar los k vecinos exactos, construye un índice HNSW por clase antes del bucle principal y consulta ese índice en cada iteración, obteniendo vecinos aproximados con un *recall* controlable. El coste por consulta pasa de $\mathcal{O}(n)$ a $\mathcal{O}(\log n)$ por instancia, transformando el cuello de botella cuadrático en un proceso de complejidad total $\mathcal{O}(n \log n \cdot m)$. Su principal limitación es que los vecinos recuperados pueden no ser exactamente los más cercanos, aunque en la práctica la diferencia

en la calidad del subconjunto de características resultante es marginal, como se analiza en detalle en el Capítulo 4.

Proto-ReliefF renuncia al tamaño del conjunto de búsqueda. En lugar de buscar vecinos entre las n instancias originales, comprime primero el conjunto de entrenamiento en $p \ll n$ prototipos mediante clustering y ejecuta ReliefF sobre ese conjunto reducido. La búsqueda sigue siendo exacta, pero el espacio sobre el que opera es mucho más pequeño, reduciendo la complejidad de $\mathcal{O}(n^2)$ a $\mathcal{O}(p^2)$. A cambio, los prototipos constituyen una representación aproximada del espacio original y pueden no preservar todas las fronteras de decisión relevantes para la selección, especialmente en regiones poco densas.

Ambas estrategias son complementarias, no equivalentes. HNSW-ReliefF conserva todas las instancias originales y aproxima la búsqueda, mientras que Proto-ReliefF mantiene la búsqueda exacta pero reduce el conjunto sobre el que opera. La elección entre una y otra depende del tamaño del conjunto de datos y de su estructura, como se discute en la Sección 3.4. La Tabla 3.1 resume las diferencias principales frente a ReliefF exacto.

	ReliefF	**HNSW-ReliefF**	**Proto-ReliefF**
Búsqueda de vecinos	Exacta	Aproximada	Exacta
Instancias consultadas	n	n	$p \ll n$
Complejidad	$\mathcal{O}(n^2 m)$	$\mathcal{O}(n \log n \cdot m)$	$\mathcal{O}(p^2 m)$
Criterio de relevancia	Original	Original	Original
Limitación principal	Escala	Recall aproximado	Pérdida de instancias

Tabla 3.1: Comparativa entre ReliefF original y las dos propuestas de este trabajo.

3.1.1 El criterio de diferencia

Antes de describir las dos propuestas conviene formalizar el criterio de relevancia que ambas comparten sin modificación. ReliefF cuantifica la diferencia entre dos instancias $x_i, x_j \in X$ en una variable f mediante la función $\text{diff}(f, x_i, x_j)$, que actúa de forma distinta según la naturaleza de la variable.

Para variables continuas, la diferencia se normaliza por el rango observado en el conjunto de entrenamiento, de modo que todas las variables contribuyan en la misma escala $[0, 1]$ con independencia de sus unidades originales:

$$\text{diff}(f, x_i, x_j) = \frac{|x_{i,f} - x_{j,f}|}{\max_f - \min_f} \tag{3.1}$$

Para variables discretas, identificadas automáticamente cuando el número de valores únicos no supera un umbral de detección (fijado en diez en ambas implementaciones), la dife-

rencia es binaria, cero si las dos instancias toman el mismo valor en esa variable, uno en caso contrario:

$$\text{diff}(f, x_i, x_j) = \begin{cases} 0 & \text{si } x_{i,f} = x_{j,f} \\ 1 & \text{en otro caso} \end{cases} \tag{3.2}$$

Esta función es el núcleo operativo de ReliefF. Acumulada sobre los vecinos más cercanos de cada instancia, estima cuánto contribuye cada variable a distinguir instancias de la misma clase de instancias de clases distintas. Un valor alto de diff en los *misses* indica que la variable separa bien las clases; un valor alto en los *hits* indica que la variable varía dentro de la misma clase, lo que penaliza su relevancia.

Las dos propuestas de este trabajo no modifican en ningún caso la Ecuación (3.1) ni la Ecuación (3.2). La eficiencia se gana modificando cómo se identifican los vecinos sobre los que se aplican estas diferencias, no el criterio de relevancia en sí. Esto garantiza que ambas propuestas preservan la capacidad de ReliefF para detectar interacciones entre variables, que depende de la estructura del vecindario pero no de la función de diferencia.

3.2 HNSW-ReliefF

HNSW-ReliefF recibe como entrada un conjunto de datos $X \in \mathbb{R}^{n \times m}$ con sus etiquetas de clase $y \in Y$ y produce como salida un vector de pesos $W \in \mathbb{R}^m$ donde cada componente refleja la relevancia estimada de la variable correspondiente. Dentro del algoritmo se diferencian tres fases, que el Algoritmo 2 resume.

1. *Construcción del índice.* Ocurre antes del bucle principal de ReliefF. El algoritmo construye un índice HNSW independiente para cada clase presente en Y e indexa únicamente las instancias de esa clase. Es una operación única, con coste $\mathcal{O}(n \log n)$.

2. *Bucle principal de actualización de pesos.* Para cada una de las n_{iter} instancias seleccionadas, el algoritmo consulta el índice de la clase propia (los k *hits*) y el de cada clase ajena (los k *misses*). Cada consulta cuesta $\mathcal{O}(\log n)$, frente al $\mathcal{O}(n^2)$ de la búsqueda exacta. Con los vecinos calculados, se aplica la Ecuación (2.2) sin modificación.

3. *Normalización y selección.* Completado el bucle, los pesos se normalizan según el número de iteraciones efectivas y se ordenan de mayor a menor; se devuelven las k variables de mayor peso.

Algoritmo 2: HNSW-ReliefF

Input: $X \in \mathbb{R}^{n \times m}$, etiquetas y, vecinos k, `iter_ratio`
Output: Vector de pesos $W \in \mathbb{R}^m$

```
Normalizar variables continuas a [0,1]
Construir un índice HNSW por clase c (búsqueda exacta solo como respaldo si la
  métrica no es compatible)
W ← 0
Muestrear n_iter = ⌊n · iter_ratio⌋ instancias de consulta
foreach instancia de consulta x_i do
    Consultar el índice de la clase y_i → k hits
    foreach clase c ≠ y_i do
        Consultar el índice de c → k misses
    Actualizar W con la Ecuación (2.2)
Normalizar W y ordenar las variables por peso
return W
```

3.2.1 Construcción del índice por clase

La decisión de construir un índice HNSW independiente para cada clase en lugar de un único índice global es la parte fundamental de la arquitectura de HNSW-ReliefF. El algoritmo necesita distinguir entre *hits* y *misses*, es decir, recuperar por separado los vecinos de la misma clase y los de cada clase distinta. Un único índice con todas las instancias devolvería en cada consulta vecinos mezclados de todas las clases, obligando a filtrarlos a posteriori hasta completar k vecinos por clase. El número de candidatos necesarios sería impredecible y potencialmente muy alto, lo que anularía la ventaja en tiempo que ofrece HNSW.

Con un índice por clase, los *hits* de x_i se recuperan consultando el índice de la clase y_i y los *misses* consultando los índices de cada clase $c \neq y_i$, obteniendo exactamente k vecinos del tipo correcto en tiempo $\mathcal{O}(\log |C_c|)$, donde $|C_c|$ es el número de instancias de la clase c. El coste total de construcción, $\mathcal{O}(n \log n)$, se distribuye entre todos los índices. Si una clase contiene una única instancia, su índice se omite, ya que no es posible recuperar vecinos de ella. Si la construcción falla por cualquier otra causa, el algoritmo recurre automáticamente a búsqueda exacta como mecanismo de respaldo, lo que garantiza robustez frente a casos especiales no contemplados.

Antes de construir los índices, las variables continuas se normalizan al rango $[0,1]$ mediante una escala de mínimo y máximo, dejando sin modificar las variables discretas, identificadas automáticamente por su número de valores únicos. Esta normalización garantiza que la métrica de distancia sea comparable entre variables con escalas distintas, de forma consistente

con el preprocesado de ReliefF original.

3.2.2 Mecanismo adaptativo

HNSW-ReliefF incorpora un mecanismo de adaptación automática que ajusta, en función del tamaño y la estructura del conjunto de datos, los parámetros del índice HNSW, la fracción de instancias empleadas como consulta y el valor efectivo de k.

Uso del índice. El índice HNSW se emplea siempre que la métrica de distancia sea compatible con él. La búsqueda exacta queda reservada como respaldo para cuando la métrica no lo permite o la construcción del índice falla (Sección 3.2.1). A diferencia de ReliefF exacto, cuyo coste cuadrático lo vuelve progresivamente inviable, HNSW-ReliefF es más rápido en todo el rango de tamaños gracias al índice y a la compilación JIT del núcleo de cómputo que puntúa cada variable. No existe un punto de corte por debajo del cual convenga la búsqueda exacta; la comparativa de escalabilidad de los tres algoritmos se analiza en el Capítulo 4.

Adicionalmente, los parámetros del índice escalan con el tamaño del conjunto para equilibrar la calidad del grafo y el coste de construcción, como se muestra en la Tabla 3.2, siguiendo la recomendación habitual en HNSW de aumentar la conectividad del grafo en conjuntos mayores para preservar el *recall* [7].

Tamaño	M	ef_construction
$n < 10000$	16	200
$10000 \leq n < 50000$	16	100
$50000 \leq n < 100000$	24	150
$n \geq 100000$	32	200

Tabla 3.2: Parámetros del índice HNSW en función del tamaño del conjunto de datos.

Fracción de iteraciones (iter_ratio). En lugar de iterar sobre las n instancias, el algoritmo muestrea aleatoriamente una fracción de ellas como instancias de consulta. Esa fracción se determina automáticamente según el tamaño del conjunto de datos, permitiendo una mayor escalabilidad. Dichos umbrales se muestran en la Tabla 3.3.

La reducción se apoya en que los pesos de ReliefF se estiman como un promedio sobre múltiples instancias. Siempre que la muestra sea representativa, la estimación converge hacia los mismos valores que con cobertura completa. Por eso, en conjuntos pequeños se mantiene el conjunto entero, mientras que en los grandes la ganancia en tiempo es sustancial y la pérdida de calidad, marginal, como muestra la Figura 3.1.

Valor adaptativo de k. Con el número de vecinos por defecto fijado a $k = 15$, el algoritmo ajusta k en función del ratio $\rho = m/n$ entre el número de variables y el de instancias,

Tamaño	iter_ratio
$n < 2000$	1,0 (100%)
$2000 \leq n < 5000$	0,8 (80%)
$5000 \leq n < 20000$	0,5 (50%)
$n \geq 20000$	0,3 (30%)

Tabla 3.3: Fracción de iteraciones utilizada en función del tamaño del conjunto.

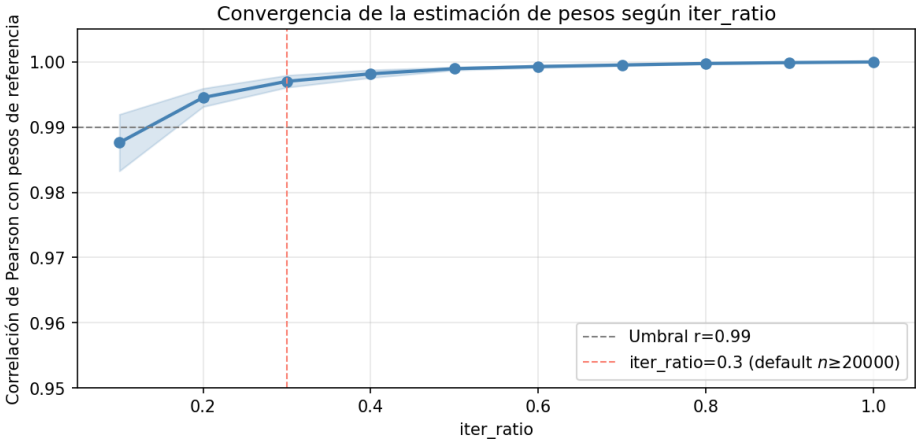


Figura 3.1: Correlación de Pearson entre los pesos estimados con distintos valores de iter_ratio y los pesos de referencia obtenidos con cobertura completa ($n = 5000$, $m = 20$, 10 repeticiones por ratio). Con iter_ratio $= 0{,}3$, valor por defecto para conjuntos grandes, la correlación media es $r = 0{,}997$, lo que confirma que el muestreo de instancias de consulta no degrada la calidad de la estimación de forma significativa.

siguiendo la función continua:

$$k_{\text{eff}} = \left\lfloor \frac{15}{1 + \rho \cdot 5{,}0} \right\rfloor, \quad k_{\text{eff}} \in [5, 15] \tag{3.3}$$

Cuando ρ es alto, el espacio es disperso y un vecindario pequeño y local resulta más representativo; cuando ρ es bajo, se mantiene un vecindario amplio para estabilizar la estimación. Por ejemplo, con $\rho = 0{,}1$ se obtiene $k_{\text{eff}} = 10$ y con $\rho = 0{,}2$ baja a $k_{\text{eff}} = 7$. Los parámetros de esta función se determinaron mediante búsqueda de hiperparámetros, cuyos resultados se presentan en el Capítulo 4.

3.2.3 Parámetros del índice HNSW

El comportamiento del índice HNSW está controlado por tres parámetros que determinan el equilibrio entre calidad de la búsqueda, velocidad de la consulta y la memoria consumida.

M es el número máximo de conexiones que cada nodo mantiene con otros nodos en cada capa del grafo. Un valor alto de M produce un grafo más denso con más caminos posibles entre nodos, aumentando de esta forma la precisión de la búsqueda pero a costa de un mayor consumo de memoria y un mayor tiempo de construcción. Un valor bajo reduce el consumo pero puede dejar regiones del espacio mal conectadas, aumentando la posibilidad de que la búsqueda acabe en un óptimo local en vez de en el global. En la implementación, M escala con el tamaño del conjunto según la Tabla 3.2, donde se expresa que se recurre a valores bajos para conjuntos medianos donde la densidad del grafo ya es suficiente y valores más altos para conjuntos muy grandes donde la conectividad necesita ser más robusta.

ef_construction controla el tamaño de la lista de candidatos en la construcción del índice. Cuando se inserta cada nuevo nodo, HNSW mantiene una cola de los *ef_construction* candidatos más prometedores para decidir con cuáles establecer conexiones. Un valor alto de este parámetro produce conexiones de mayor calidad y un grafo más preciso, pero aumenta el tiempo de construcción de manera lineal. La implementación utiliza valores entre 100 y 200 según el tamaño del conjunto, priorizando la calidad del índice en conjuntos grandes donde cada consulta se amortiza sobre más iteraciones.

ef_search controla el tamaño de la lista de candidatos durante la consulta, a diferencia de los dos anteriores, este parámetro puede ajustarse sin la necesidad de reconstruir el índice, permitiendo modificar el balance precisión-velocidad en tiempo de ejecución. En la implementación, si no se especifica un valor explícito, se calcula automáticamente como $\max(2k, 50)$, garantizando que la lista de candidatos sea siempre al menos el doble del número de vecinos valorados. Aumentar ef_search mejora el *recall* a costa de mayor tiempo de consulta, por otro lado, reducirlo acelera la búsqueda pero puede degradar la calidad de los vecinos recuperados.

La interacción entre estos tres parámetros define el comportamiento global del índice. Por un lado, M y ef_construction determinan la calidad del grafo construido; por otro, ef_search determina cuánto de ese potencial se aprovecha en cada consulta. En el contexto de este trabajo, donde no se necesita una precisión perfecta, sino suficientemente buena para que la estimación de las puntuaciones sea fiable, los valores por defecto de los tres parámetros ofrecen un buen equilibrio sin necesidad de ajuste manual.

3.2.4 Análisis de complejidad

La sustitución de la búsqueda exacta por el índice HNSW transforma la complejidad cuadrática inicial de ReliefF de forma significativa. A continuación se detalla el impacto, distin-

guiendo entre el coste de construcción del índice, el del bucle principal y el de memoria:

- *Construcción del índice.* Construir un índice HNSW sobre n_c instancias cuesta $\mathcal{O}(n_c \log n_c)$ por clase. Sumando todas las clases, el coste total es $\mathcal{O}(n \log n)$, ya que $\sum_c n_c = n$. Esta operación se realiza una única vez antes del bucle principal, por lo que su coste queda amortizado sobre las n_{iter} iteraciones posteriores.

- *Bucle principal.* Cada iteración requiere una consulta al índice de la clase propia y una por cada clase ajena. Con $|\mathcal{C}|$ clases, el coste por iteración es $\mathcal{O}(|\mathcal{C}| \cdot \log n)$, de modo que sobre $n_{\text{iter}} = n \cdot \texttt{iter\_ratio}$ iteraciones y m variables el coste total del bucle es $\mathcal{O}(n \cdot \texttt{iter\_ratio} \cdot |\mathcal{C}| \cdot \log n \cdot m)$. En el caso binario con $\texttt{iter\_ratio} = 1$ se reduce a $\mathcal{O}(n \log n \cdot m)$, frente al $\mathcal{O}(n^2 \cdot m)$ de ReliefF; la mejora en tiempo de ejecución es sustancial y se cuantifica empíricamente en el Capítulo 4.

- *Memoria.* El índice HNSW almacena hasta M conexiones por capa y nodo, con un consumo de $\mathcal{O}(n \cdot M)$ que crece linealmente en el número de instancias. La matriz de distancias de ReliefF, de tamaño $\mathcal{O}(n^2)$, resulta inviable mantener en memoria a partir de cierto umbral, como se discutió anteriormente.

La comparativa conjunta de complejidades con Proto-ReliefF se recoge en la Sección 3.3.4, una vez presentada la segunda propuesta.

3.2.5 Implementación eficiente: compilación JIT con Numba

El análisis de complejidad de la sección anterior muestra que el cuello de botella de HNSW-ReliefF, una vez sustituida la búsqueda cuadrática por el índice HNSW, se desplaza al bucle interno de acumulación de diferencias. Para cada instancia de consulta y cada vecino recuperado, el algoritmo debe calcular $\text{diff}(f, x_i, x_j)$ para las m variables y acumular el resultado en el vector de pesos W. Con n_{iter} instancias, k vecinos por clase y $|\mathcal{C}|$ clases, este bucle ejecuta del orden de $n_{\text{iter}} \cdot k \cdot |\mathcal{C}| \cdot m$ operaciones aritméticas elementales (potencialmente cientos de millones para conjuntos de tamaño moderado) en Python puro. La sobrecarga del intérprete hace que esta fase domine el tiempo total incluso cuando la búsqueda HNSW es rápida.

La solución adoptada es compilar el núcleo de puntuación mediante Numba `@njit`, que transforma el código Python anotado en instrucciones máquina optimizadas en la primera llamada. El decorador utilizado combina `fastmath=True`, que permite al compilador reordenar operaciones de coma flotante para maximizar el uso de unidades vectoriales SIMD, con `parallel=True`, que distribuye el bucle externo (el que recorre las instancias de consulta) entre todos los núcleos disponibles mediante OpenMP.

La restricción fundamental del modo `@njit` es que únicamente admite arrays NumPy y tipos numéricos escalares: no es posible usar listas de Python, diccionarios, objetos de scikit-

learn ni ninguna otra abstracción del intérprete. Por este motivo, la implementación separa de forma explícita la lógica de alto nivel (construcción del índice, muestreo de instancias de consulta, extracción de vecinos), que permanece en Python, de la lógica de bajo nivel (acumulación de diferencias ponderadas, cálculo de W), que se transfiere al kernel JIT. La interfaz entre ambas partes es un array denso de índices de vecinos de forma $(n_{\text{iter}}, k_{\max})$ pre-rellenado con -1 para las posiciones vacías, que el kernel ignora con una comprobación de guardia.

El coste de esta arquitectura es el tiempo de compilación en la primera llamada: Numba necesita entre 4 y 5 segundos para compilar el kernel la primera vez que se ejecuta con una signatura de tipos dada. Este *overhead* de calentamiento (*warm-up*) es fijo e independiente del tamaño del dataset, y queda completamente amortizado desde la segunda ejecución en la misma sesión de Python. Para uso en *pipelines* de evaluación donde el selector se instancia múltiples veces, la compilación se produce una única vez por tipo de array (`float32`, `int32`) y se reutiliza en todas las llamadas posteriores.

El beneficio es sustancial: el bucle de acumulación compilado por Numba con paralelismo sobre c núcleos reduce el tiempo de la puntuación aproximadamente en un factor de $c \cdot 10$ respecto al mismo bucle en Python puro para arrays `float32` de tamaño típico. Combinado con el índice HNSW, HNSW-ReliefF consigue que el tiempo total de ejecución escale con $\mathcal{O}(n \log n)$ no solo en términos de complejidad teórica sino también en constantes de tiempo medidas empíricamente.

3.2.6 Decisiones de diseño de HNSW-ReliefF

El diseño de HNSW-ReliefF descansa sobre tres decisiones arquitectónicas que no son evidentes y que determinan su comportamiento en la práctica. La estructura de índice elegida, su organización por clases y la forma de reducir el coste del bucle principal son justificadas a continuación.

La primera es la elección de HNSW frente a otras estructuras de índice espacial. Los k-d trees [27] funcionan bien en dimensionalidad baja, pero degeneran cuando m supera aproximadamente veinte dimensiones. En ese régimen, la *maldición de la dimensionalidad* vuelve casi equidistantes a todas las instancias, lo que obliga a explorar prácticamente todo el árbol en cada consulta y recupera la complejidad $\mathcal{O}(n)$ del caso sin índice. LSH (*Locality-Sensitive Hashing*) ofrece garantías probabilísticas, pero exige ajustar con cuidado el número de tablas y las funciones de *hash* para cada distribución, y su *recall* efectivo puede ser bajo en dimensionalidad moderada si el número de tablas no es muy grande. HNSW, en cambio, mantiene un coste $\mathcal{O}(\log n)$ por consulta con independencia de la dimensionalidad gracias a su grafo navegable jerárquico, y supera de forma sistemática a FLANN, Annoy, FAISS y otras bibliotecas ANN en los *benchmarks* de referencia [7]. Su único inconveniente, un mayor consumo de memoria, es asumible, ya que el índice almacena solo M conexiones por nodo y no la matriz

de distancias completa.

La segunda decisión es construir un índice independiente por clase en lugar de uno global. ReliefF necesita distinguir, para cada instancia de consulta x_i, entre los vecinos de su misma clase (*hits*) y los de cada clase distinta (*misses*). Un único índice con todas las instancias devolvería vecinos mezclados en cada consulta, obligando a filtrar los resultados hasta reunir k vecinos de cada tipo. El número de candidatos necesarios para garantizar k vecinos de la clase minoritaria tras ese filtrado es imprevisible, de decenas a miles según la distribución de clases, lo que anularía la ventaja en velocidad. Con un índice por clase, los *hits* de x_i se obtienen consultando directamente el índice de la clase y_i, que devuelve exactamente k vecinos del tipo correcto en $\mathcal{O}(\log |C_{y_i}|)$, y el coste de construcción se reparte de forma proporcional, con $\sum_c \mathcal{O}(n_c \log n_c) \leq \mathcal{O}(n \log n)$.

La tercera decisión afecta a la forma de reducir el coste del bucle principal, donde se opta por muestrear instancias de consulta (`iter_ratio`) y no variables. Muestrear variables modificaría el estimador, porque actualizar en cada iteración solo un subconjunto aleatorio de ellas haría que los pesos se estimaran con distinto número de muestras, lo que introduce una varianza heterogénea que sesga la ordenación final. Muestrear instancias, en cambio, reduce el número de consultas al índice sin alterar la función estimada, ya que los pesos siguen siendo un promedio sobre instancias elegidas al azar que, mientras la muestra sea representativa, converge hacia el mismo valor que con cobertura completa. La Figura 3.1 lo confirma empíricamente, con una correlación de Pearson de $r = 0{,}997$ entre los pesos muestreados con $\texttt{iter\_ratio} = 0{,}3$ y los de referencia, de modo que la reducción del 70% en tiempo de ejecución no degrada la calidad de la selección.

3.3 Proto-ReliefF

Proto-ReliefF recibe como entrada un conjunto de datos $X \in \mathbb{R}^{n \times m}$ con sus etiquetas de clase y y produce como salida un vector de pesos $W \in \mathbb{R}^m$. A diferencia de HNSW-ReliefF, la búsqueda de vecinos sigue siendo exacta; lo que cambia es el conjunto sobre el que se realiza. En lugar de operar sobre las n instancias originales, Proto-ReliefF las comprime en $p \ll n$ prototipos representativos y ejecuta ReliefF sobre ellos. La ganancia de eficiencia proviene, por tanto, de reducir el tamaño del problema, no de relajar la exactitud de la búsqueda. El Algoritmo 3 resume el proceso, que se articula en tres fases.

1. *Generación de prototipos.* Antes de estimar la relevancia, cada clase se comprime por separado mediante *clustering*, obteniendo un conjunto reducido de representantes anclados a instancias reales.

2. *Puntuación ponderada.* Sobre los prototipos se aplica el criterio de ReliefF, sustituyendo el promedio uniforme de los vecinos por una ponderación que prima a los más cercanos.

3. *Normalización y selección.* Los pesos resultantes se normalizan y se ordenan de mayor a menor, devolviendo las k variables de mayor relevancia.

Algoritmo 3: Proto-ReliefF

Input: $X \in \mathbb{R}^{n \times m}$, etiquetas y, compresión σ_{eff}, vecinos k
Output: Vector de pesos $W \in \mathbb{R}^m$

```
Normalizar variables continuas y detectar las discretas
foreach clase c do
    Comprimir X_c en p_c prototipos mediante clustering
    Aplicar snap-to-grid: cada centroide → instancia real más cercana
W ← 0
foreach clase c do
    Acumular en W la contribución de ReliefF sobre los prototipos con ponderación
     softmax (Ec. (3.9))
Normalizar W y ordenar las variables por peso
return W
```

3.3.1 Generación de prototipos

Para que la compresión preserve la geometría que ReliefF necesita, las variables continuas se normalizan previamente con una escala robusta frente a valores atípicos. En lugar de reescalar por el mínimo y el máximo, muy sensibles a los extremos, cada variable se centra en su mediana $\tilde{v}$ y se divide por su rango intercuartílico:

$$v' = \frac{v - \tilde{v}}{\text{IQR}(v)}, \qquad \text{IQR}(v) = Q_{75} - Q_{25} \tag{3.4}$$

Los valores resultantes se recortan al intervalo $[-5, 5]$ para acotar el efecto de los atípicos más extremos, mientras que las variables discretas, identificadas automáticamente por tener un número reducido de valores únicos, se mantienen intactas para no alterar su estructura categórica.

Sobre los datos ya normalizados, cada clase se comprime por separado. Para la clase c se agrupan sus instancias $X_c = \{x_i : y_i = c\}$ mediante *clustering* y se obtienen p_c centroides, en un número proporcional al tamaño de la clase:

$$p_c = \max(1,\ \min(200,\ \lfloor n_c \cdot \sigma_{\text{eff}} \rfloor)) \tag{3.5}$$

donde n_c es el número de instancias de la clase y σ_{eff} el factor de compresión efectivo. El límite inferior garantiza al menos un prototipo por clase, de modo que ninguna queda sin representar

por pequeña que sea, y el superior acota el coste cuando la clase es muy numerosa. Si la clase tiene menos instancias que prototipos solicitados, se emplean todas directamente.

Los centroides así obtenidos plantean un problema. Al calcularse como medias de los puntos de su grupo, no coinciden con ninguna instancia real y pueden tomar valores fuera del dominio de las variables, como un $0{,}5$ en una variable binaria, lo que falsearía las distancias. Para evitarlo se aplica un paso de *snap-to-grid*, que reemplaza cada centroide abstracto c_j por la instancia real de su clase más próxima:

$$p_j^* = \arg\min_{x \in X_c} \|x - c_j\|_2 \tag{3.6}$$

De esta forma todos los prototipos son instancias legítimas del conjunto y conservan valores válidos en cada variable, también en las discretas.

El factor de compresión no es fijo, sino que se adapta a la densidad del espacio. Cuando hay muchas variables por instancia el espacio está poco poblado y conviene retener más representantes, de modo que σ_{eff} crece con el ratio $\rho = m/n$ siguiendo una función logarítmica acotada:

$$\sigma_{\text{eff}} = \min\left(\max\left(0{,}10,\, 0{,}10 \cdot \left(1 + \log \frac{\rho}{0{,}05}\right)\right),\, 0{,}25\right) \tag{3.7}$$

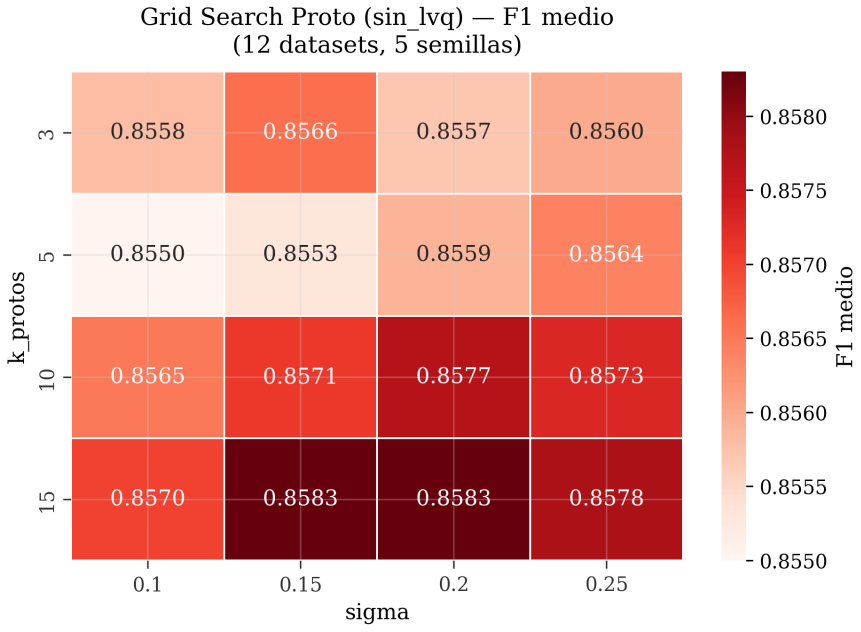


Figura 3.2: Mapa de calor de F1 de Proto-ReliefF en función del número de prototipos y del factor de compresión σ (sin refinamiento LVQ). La sensibilidad a σ es menor que la del número de prototipos; el valor $\sigma = 0{,}15$ ofrece el mejor compromiso entre coste y calidad.

La Figura 3.2 justifica los límites de ese intervalo. Por debajo de $\sigma = 0{,}10$ quedan tan pocos prototipos que la variabilidad local se pierde y ReliefF deja de captar las interacciones entre variables; por encima de $\sigma = 0{,}25$ la compresión deja de aportar una reducción de coste apreciable. El valor por defecto $\sigma = 0{,}15$ ofrece el mejor compromiso entre calidad y coste, como confirma la búsqueda de hiperparámetros del Capítulo 4.

3.3.2 Ponderación adaptativa de prototipos

La diferencia esencial entre puntuar sobre prototipos y hacerlo sobre las instancias originales está en la escala del vecindario. ReliefF estándar promedia por igual las contribuciones de los k vecinos más cercanos, algo razonable cuando todos están a distancias comparables. Sobre un conjunto comprimido, en cambio, el vecino más próximo y el más lejano pueden encontrarse a distancias muy dispares, de modo que un promedio uniforme otorgaría el mismo peso a un prototipo muy representativo que a otro alejado, diluyendo la señal.

Para corregirlo, Proto-ReliefF sustituye el promedio uniforme por una ponderación *softmax* de distancia inversa, que concede más influencia a los prototipos cercanos:

$$w_j = \frac{\exp(-\gamma \cdot d(x_i, p_j))}{\sum_{l=1}^{k} \exp(-\gamma \cdot d(x_i, p_l))} \tag{3.8}$$

donde $d(x_i, p_j)$ es la distancia euclídea entre la instancia x_i y el prototipo p_j, y γ la temperatura que regula la concentración de la distribución. Un valor alto de γ concentra casi todo el peso en el prototipo más cercano, comportándose como un 1-NN ponderado, mientras que $\gamma \to 0$ recupera el promedio uniforme de ReliefF estándar. Por defecto γ se fija de forma automática como el inverso de la desviación típica de las distancias del vecindario, $\hat{\gamma} = 1/\sigma_d$, lo que normaliza la ponderación según la dispersión local y la hace invariante a la escala absoluta de las distancias.

Con estos pesos, la actualización de cada variable pasa de un promedio a una suma ponderada:

$$W(f) \mathrel{+}= \sum_{j=1}^{k} w_j \cdot \text{diff}(f, x_i, p_j) \tag{3.9}$$

con diff la función de diferencia de las Ecuaciones (3.1) y (3.2). La ponderación se aplica por separado a los *hits* y a los *misses* de cada clase, preservando la estructura multiclase del criterio original recogido en la Ecuación (2.2).

3.3.3 Parámetros de compresión

El comportamiento de Proto-ReliefF queda determinado por unos pocos parámetros, recogidos en la Tabla 3.4. Los dos más influyentes son el factor de compresión σ, que fija el grado de reducción y, con él, el tiempo de cómputo, y el número de vecinos k, que controla la calidad de la estimación. La temperatura γ ajusta la ponderación descrita en la Sección 3.3.2, y el indicador `use_lvq` habilita un refinamiento opcional que, como se justifica más adelante, permanece desactivado por defecto.

Parámetro	Defecto	Rango	Efecto
σ	adaptativo	$[0{,}10,\ 0{,}25]$	Grado de compresión
k	10	$[1,\ p_{\min}]$	Vecinos en la puntuación
γ	automático	> 0	Concentración de la ponderación
use_lvq	False	booleano	Refinamiento de prototipos

Tabla 3.4: Parámetros principales de Proto-ReliefF con sus valores por defecto y su efecto.

Al igual que en HNSW-ReliefF (Sección 3.2.2), el número de vecinos no es rígido, sino que se adapta al tamaño de cada clase, descendiendo hasta tres vecinos en las clases muy pequeñas y subiendo hasta quince en las grandes. Así, la puntuación se mantiene estable aunque el número de prototipos por clase varíe notablemente. El impacto de cada parámetro sobre la calidad final se cuantifica en la búsqueda de hiperparámetros del Capítulo 4 (Sección 4.2).

Este mismo mecanismo absorbe el desequilibrio entre clases, una situación delicada para cualquier método basado en vecindarios. Como la compresión se aplica clase a clase y garantiza al menos un prototipo por clase, la representación relativa de las minoritarias se preserva: una clase con veinte instancias y otra con dos mil se comprimen a la misma tasa. Cuando el desequilibrio es acusado, el número de vecinos se reduce ligeramente para que la clase mayoritaria no domine la estimación, y la ponderación por probabilidad a priori heredada del criterio de ReliefF, presentado en la Ecuación (2.2), modera la contribución de las clases más frecuentes. El resultado es un comportamiento robusto frente a distribuciones de clase muy dispares sin necesidad de remuestreo previo.

3.3.4 Análisis de complejidad

La compresión a $p = \sum_c p_c$ prototipos transforma la complejidad de ReliefF. La normalización inicial es lineal, $\mathcal{O}(n \cdot m)$, y la generación de prototipos añade un coste también lineal en n para un número fijo de representantes. El término dominante es la puntuación exacta sobre los prototipos, $\mathcal{O}(p^2 \cdot m)$, muy inferior al $\mathcal{O}(n^2 \cdot m)$ original: para un conjunto de $n = 50\,000$ instancias con $\sigma_{\text{eff}} = 0{,}10$ se obtienen unos $p \approx 400$ prototipos, lo que reduce

el coste de la búsqueda en más de dos órdenes de magnitud. En memoria la mejora es aún más marcada: frente a la matriz de distancias $\mathcal{O}(n^2)$ de ReliefF exacto, basta con almacenar los prototipos, $\mathcal{O}(p \cdot m)$, lo que elimina el límite de memoria que aparece a partir de $n > 10^4$.

La Tabla 3.5 reúne las complejidades de las tres implementaciones. La comparación deja claro que HNSW-ReliefF y Proto-ReliefF no compiten, sino que se complementan: la primera conserva las n instancias y rebaja la búsqueda a $\mathcal{O}(n \log n)$ con memoria lineal, idónea cuando interesa preservar toda la estructura local; la segunda comprime el conjunto a $p \ll n$ prototipos con búsqueda exacta sobre ellos, ventajosa cuando el tamaño o la memoria son el factor limitante. La Figura 3.3 confirma empíricamente estas complejidades: las rectas de regresión en escala log-log arrojan un exponente $\alpha \approx 2$ para ReliefF y $\alpha < 1$ para ambas propuestas.

	ReliefF	**HNSW-ReliefF**	**Proto-ReliefF**
Preprocesado	$\mathcal{O}(1)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \cdot m)$
Generación repr.	No aplica	No aplica	$\mathcal{O}(n \cdot p \cdot m)$
Búsqueda	$\mathcal{O}(n^2 m)$	$\mathcal{O}(n \log n \cdot m)$	$\mathcal{O}(p^2 m)$
Memoria	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot M)$	$\mathcal{O}(p \cdot m)$

Tabla 3.5: Complejidades temporal y espacial de las tres implementaciones. $p \ll n$ es el número de prototipos en Proto-ReliefF y M el parámetro de conectividad del índice HNSW.

3.3.5 Implementación eficiente

El análisis anterior describe el coste asintótico, pero alcanzarlo en la práctica exige aprovechar dos propiedades de la fase de puntuación. La primera es que las clases se puntúan de forma independiente. La contribución de cada clase al vector de pesos se calcula leyendo únicamente sus prototipos, sin escrituras compartidas, de modo que no existen dependencias entre clases ni condiciones de carrera. Esto convierte la puntuación en una tarea trivialmente paralelizable, repartida entre los núcleos disponibles asignando una clase por tarea. El reparto solo compensa cuando hay varias clases y suficientes instancias; por debajo de ese umbral el coste de coordinar los procesos supera al cómputo, y la implementación revierte de forma automática a una ejecución secuencial.

La segunda es el uso de precisión mixta para contener el consumo de memoria. El cálculo de distancias entre instancias y prototipos, que es la operación más voluminosa, se realiza en una representación numérica de menor tamaño, mientras que la acumulación de los pesos, donde el error numérico sí importa, se mantiene en precisión completa. De este modo se reduce a la mitad el ancho de banda de memoria de la parte más pesada sin sacrificar exactitud en el resultado. La combinación de ambas técnicas permite a Proto-ReliefF escalar a conjuntos que

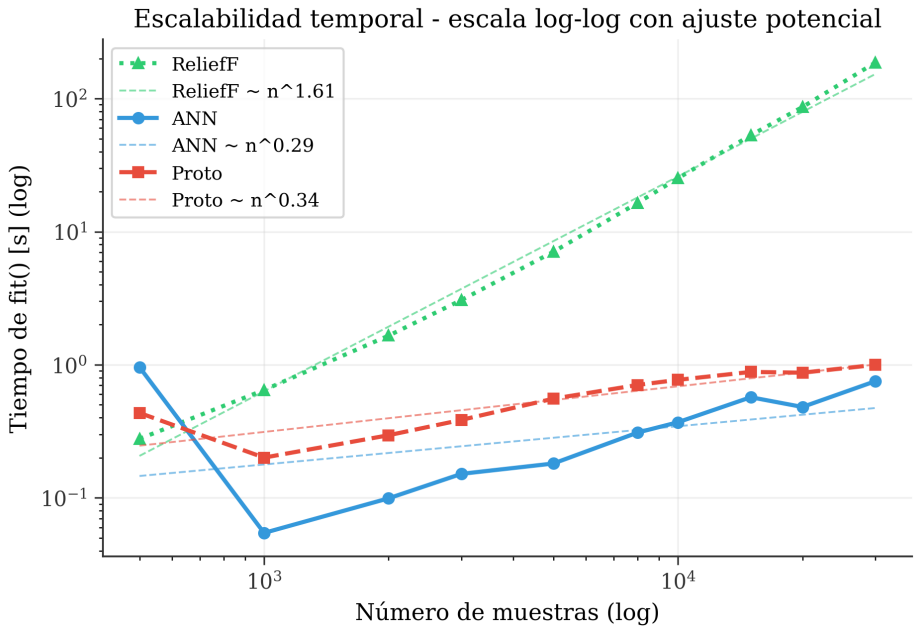


Figura 3.3: Escalabilidad temporal en escala log-log de los tres algoritmos. Las líneas de regresión estiman el exponente α de la ley de potencia $t \propto n^{\alpha}$: ReliefF muestra $\alpha \approx 2$ ($\mathcal{O}(n^2)$); HNSW-ReliefF y Proto-ReliefF muestran $\alpha < 1$, confirmando la ganancia de complejidad (Sección 4.3).

desbordarían en memoria a HNSW-ReliefF, conservando un tiempo de ejecución competitivo, como se comprueba en el Capítulo 4.

3.3.6 Decisiones de diseño de Proto-ReliefF

Como HNSW-ReliefF, Proto-ReliefF se apoya en varias decisiones de diseño que conviene justificar, relativas al algoritmo de *clustering*, al *snap-to-grid*, al factor de compresión adaptativo y al refinamiento opcional de los prototipos.

La primera es emplear MiniBatchKMeans como algoritmo de *clustering*. Una alternativa natural sería *K-Medoids*, que por construcción usa instancias reales como centros y haría innecesario el *snap-to-grid*, pero su coste por iteración es cuadrático en el número de prototipos y lo vuelve inviable para conjuntos grandes. MiniBatchKMeans es varios órdenes de magnitud más rápido, y el *snap-to-grid* posterior recupera la propiedad de centros reales a un coste lineal. La combinación de ambos puede entenderse como una aproximación eficiente a *K-Medoids* en dos pasos, uno que localiza los centros con rapidez y otro que los proyecta a la instancia real más próxima.

La segunda decisión, el *snap-to-grid*, no es solo una cuestión de velocidad, sino de corrección. Los centroides de MiniBatchKMeans son medias aritméticas y, en variables discretas o

binarias, producen valores imposibles. Por ejemplo, la media de $\{0, 0, 1, 1\}$ es $0{,}5$, lo que falsearía la función de diferencia de la Ecuación (3.2). Al anclar cada prototipo a una instancia real, el *snap-to-grid* garantiza valores válidos en todas las variables y preserva la corrección del cálculo de relevancia con cualquier mezcla de tipos.

La tercera es definir la compresión con un *sigma* adaptativo en lugar de un valor fijo. Un número fijo de prototipos produciría tasas de compresión muy dispares según el tamaño del conjunto, agresivas en exceso para los más grandes. Expresar σ_{eff} como fracción del tamaño de cada clase mantiene la compresión siempre entre el 10 % y el 25 % de las instancias, y la corrección logarítmica de la Ecuación (3.7) eleva esa fracción cuando el espacio es disperso, donde se necesitan más representantes.

La última decisión es mantener desactivado por defecto el refinamiento LVQ, un ajuste supervisado que reposiciona los prototipos para separar mejor las clases. En los experimentos de este trabajo no mejora la calidad de forma consistente y en ocasiones la degrada ligeramente, porque entra en conflicto con el *snap-to-grid*. Al desplazar los prototipos a posiciones intermedias deja de garantizar que sean instancias reales y se pierde la corrección en variables discretas. Ambos mecanismos son, pues, parcialmente incompatibles, de modo que activar LVQ obligaría a renunciar al *snap-to-grid*; por ello se ofrece como opción y no como comportamiento por defecto.

3.4 Comparativa y guía de uso

Las dos propuestas atacan el mismo cuello de botella desde ángulos complementarios y presentan perfiles de compromiso distintos. La Tabla 3.6 amplía la comparativa inicial de la Sección 3.1 incorporando las dimensiones relevantes para elegir entre una y otra propuesta.

	ReliefF	**HNSW-ReliefF**	**Proto-ReliefF**
Búsqueda de vecinos	Exacta	Aproximada	Exacta sobre P
Instancias consultadas	n	n	$p \ll n$
Complejidad temporal	$\mathcal{O}(n^2 m)$	$\mathcal{O}(n \log n \cdot m)$	$\mathcal{O}(p^2 m)$
Complejidad espacial	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot M)$	$\mathcal{O}(p \cdot m)$
Instancias preservadas	Todas	Todas	Solo prototipos
Garantía de recuperación	100%	$\approx$ 95–99% recall	100% sobre P
Alta dimensionalidad	Inviable	Eficiente	Eficiente
Fronteras de decisión sutiles	Completo	Completo	Depende de la compresión
Limitación principal	Escala cuadrática	Recall aproximado	Fidelidad de la compresión

Tabla 3.6: Comparativa completa entre las tres implementaciones según criterios de complejidad, precisión y condiciones de aplicabilidad.

HNSW-ReliefF es preferible cuando la distribución del espacio es compleja y la estructura local es informativa. Al preservar todas las instancias originales no pierde información sobre fronteras de decisión en regiones poco densas, y la aproximación introducida por HNSW raramente afecta a la calidad de la estimación de relevancia, como se verifica empíricamente en el Capítulo 4. Es especialmente adecuado para conjuntos con n entre 10^4 y 10^6 y dimensionalidad moderada o alta.

Mientras, por otro lado, recurrir a Proto-ReliefF es preferible cuando n es tan grande que construir un índice HNSW resulta prohibitivo en memoria o tiempo, o cuando la distribución por clases puede representarse fielmente con pocos prototipos. Esto ocurre habitualmente en conjuntos con clases compactas y bien separadas, donde el clustering captura la mayor parte de la variabilidad relevante. Su limitación es que la compresión puede eliminar instancias que delimitan fronteras de decisión sutiles, especialmente en distribuciones multimodales o con clases muy solapadas.

En la práctica, para $n < 10^4$ ReliefF exacto sigue siendo viable y debe preferirse por su exactitud y ausencia de hiperparámetros adicionales. Para $10^4 \leq n \leq 10^5$, HNSW-ReliefF ofrece la mejor relación entre velocidad y calidad de la selección. Para $n > 10^5$ o cuando la memoria es el factor limitante, Proto-ReliefF es la alternativa natural.

3.4.1 Interacción entre hiperparámetros

Ambas propuestas exponen hiperparámetros cuya interacción determina el equilibrio entre velocidad y calidad de la selección. Comprender estas interacciones es necesario para ajustar cada algoritmo correctamente en casos de uso que se aparten de los valores por defecto.

En HNSW-ReliefF, los tres parámetros del índice (M, `ef_construction` y `ef_search`) actúan sobre dimensiones distintas y son casi ortogonales entre sí. M y `ef_construction` determinan la calidad estructural del grafo en la fase de construcción. Una vez que el índice está construido, no es posible mejorar su calidad sin reconstruirlo. `ef_search` actúa en la fase de consulta y puede modificarse en cualquier momento sin reconstrucción, lo que permite ajustar el trade-off precisión/velocidad durante la ejecución. El parámetro `iter_ratio` actúa sobre una dimensión completamente independiente, reduce el número de consultas al índice sin cambiar la calidad de cada consulta individual. En consecuencia, para maximizar la velocidad se puede reducir `iter_ratio` primero, con impacto mínimo en la calidad como muestra la Figura 3.1, y ajustar `ef_search` como segundo paso si se necesita una reducción adicional en tiempo.

En Proto-ReliefF, los parámetros σ_{eff} y k interactúan de forma no trivial porque σ_{eff} controla el número de prototipos por clase y k controla cuántos de esos prototipos se consultan durante la puntuación. Si σ_{eff} es bajo, p_c puede ser pequeño, y pedir $k > p_c$ vecinos es imposible. El mecanismo adaptativo de k_{protos} evita este problema ajustando k en función del

tamaño mínimo de clase antes de calcular los prototipos; sin embargo, si se especifican valores manuales de σ y k que entren en conflicto, la implementación ajusta silenciosamente k a $\min(k, p_c)$. Una segunda interacción relevante es entre σ_{eff} y `min_cluster_size`: si σ_{eff} es alto y `min_cluster_size` también, muchos clústeres pequeños pueden ser descartados, reduciendo p_c efectivo por debajo del valor nominal. En distribuciones con alta varianza intraclase (clases muy dispersas), esta interacción puede degradar la representación; en esos casos, reducir `min_cluster_size` a 2 o 3 preserva más prototipos a costa de representantes menos estables.

El parámetro γ de la ponderación softmax interactúa con la escala de las distancias: valores altos de γ concentran la masa en el prototipo más cercano (equivalente a 1-NN ponderado), valores bajos recuperan el promedio uniforme de ReliefF estándar. Con $\gamma =$ `auto`, el algoritmo estima $\hat{\gamma} = 1/\sigma_d$ donde σ_d es la desviación típica de las distancias en el vecindario, normalizando la ponderación por la dispersión local. Si la compresión es agresiva (σ_{eff} bajo, pocos prototipos) las distancias al vecindario son más variables y $\hat{\gamma}$ es menor, produciendo una ponderación más uniforme que compensa la menor resolución del conjunto de prototipos.

3.4.2 Limitaciones y extensiones posibles

Las dos propuestas resuelven el cuello de botella cuadrático de ReliefF pero introducen limitaciones propias que conviene documentar para orientar su uso y señalar direcciones de trabajo futuro.

La limitación principal de HNSW-ReliefF es la degradación del *recall* en distribuciones particularmente difíciles. El grafo HNSW se construye con conectividad fija (M conexiones por nodo) y si la distribución del espacio es muy heterogénea, con regiones de alta densidad alternando con regiones muy dispersas, las regiones dispersas pueden quedar mal conectadas, aumentando la probabilidad de que la búsqueda quede atrapada en un óptimo local. Este problema es bien conocido en la literatura de ANN [28] y puede mitigarse aumentando M y `ef_construction`, a costa de mayor tiempo de construcción y memoria. Una extensión natural sería un mecanismo de *recall* adaptativo que detecte distribuciones difíciles (por ejemplo, midiendo la varianza del número de vecinos distintos recuperados en consultas repetidas) y ajuste `ef_search` automáticamente para garantizar un *recall* mínimo.

La limitación principal de Proto-ReliefF es la dependencia de la calidad del clustering. MiniBatchKMeans asume implícitamente que las clases tienen estructura compacta y aproximadamente unimodal; cuando una clase tiene una distribución multimodal (dos o más grupos separados de instancias), el clustering puede producir prototipos que representan la media de modos distintos y no representan bien ninguno de ellos. El *snap-to-grid* mitiga parcialmente este problema proyectando los centroides a instancias reales, pero si los modos están muy separados, el centroide abstracto puede estar equidistante de ambos modos y el *snap* produce

un resultado aleatorio. Una extensión posible es usar clustering jerárquico o modelos de mezcla gaussiana (GMM) para detectar la multimodalidad y ajustar el número de prototipos por modo, a costa de mayor complejidad en la fase de generación.

Ambas propuestas comparten la limitación de no soportar aprendizaje incremental. Cuando llegan nuevas instancias, el índice HNSW debe reconstruirse y los prototipos de Proto-ReliefF deben recalcularse. Si bien *hnswlib* soporta inserción de nuevos nodos sin reconstrucción completa, la actualización de los pesos de ReliefF requeriría reiniciar el bucle de puntuación para garantizar consistencia. Para aplicaciones de selección de características en *streaming*, donde los datos llegan en flujo continuo, ambas propuestas deberían combinarse con estrategias de actualización incremental de los pesos, como ventanas deslizantes o descuentos exponenciales, que están fuera del alcance de este trabajo pero constituyen una línea de extensión directa.

Capítulo 4

Experimentación y resultados

Este capítulo valida empíricamente las dos propuestas del Capítulo 3. El hilo argumental sigue el orden de la contribución, primero se demuestra que HNSW-ReliefF y Proto-ReliefF son mucho más rápidos que ReliefF exacto (Sección 4.3), luego que escalan a tamaños donde ReliefF es directamente inejecutable (Sección 4.4), y que esa reducción de cómputo se traduce en una menor huella de carbono (Sección 4.5). Establecida la ventaja en eficiencia, la Sección 4.6 verifica que no se paga con una pérdida de calidad de selección y la Sección 4.7 descompone la aceleración para identificar la contribución de cada componente del diseño.

4.1 Configuración experimental

Antes de presentar los resultados, esta sección establece el marco experimental común a todos los análisis del capítulo, desde los conjuntos de datos empleados y el protocolo de evaluación hasta el plan de experimentación y el entorno de ejecución.

4.1.1 Conjuntos de datos

La experimentación combina conjuntos de tres familias. Doce conjuntos de tamaño pequeño y medio (Tabla 4.1), que mezclan datos reales de referencia y datos sintéticos con *ground truth* conocido de características relevantes, sirven para los análisis de calidad e hiperparámetros. Conjuntos sintéticos generados con `make_classification` y tamaño creciente hasta diez millones de instancias sustentan el estudio de escalabilidad. Finalmente, tres conjuntos reales masivos (CoverType, SUSY y HEPMASS) validan la viabilidad a gran escala sobre datos no sintéticos.

Para el estudio de escalabilidad a gran escala se emplean los conjuntos reales recogidos en la Tabla 4.2, que cubren desde cientos de miles hasta varios millones de instancias. Al ser datos reales, no disponen de *ground truth* de relevancia, por lo que sobre ellos se mide el tiempo de ajuste y el F1 macro con un clasificador posterior, no el acierto en la recuperación de variables.

Tabla 4.1: Conjuntos de datos de tamaño pequeño y medio empleados en los análisis de calidad e hiperparámetros. Los marcados con † son sintéticos con *ground truth* de características relevantes conocido.

Nombre	n	m	Clases	Origen	Relevantes
Breast_Cancer	569	30	2	UCI / sklearn	desconocido
Wine	178	13	2	UCI / sklearn	desconocido
Digits	1797	64	2	NIST / sklearn	desconocido
Corral†	800	20	2	sintético	6 (AND/OR)
CorrAL100†	1600	100	2	sintético	4 + 1 correlada + 95 ruido
XOR†	1000	10	2	sintético	2
Moons	1000	2	2	sklearn	2
Circles	1000	2	2	sklearn	2
AltaDim†	500	100	2	sintético	10 de 100
Desbalanceado†	1000	20	2	sintético	15 (ratio 70/30)
Ruidoso†	800	25	2	sintético	10 (20 % ruido etiquetas)
Grande†	5000	30	2	sintético	20 de 30

Tabla 4.2: Conjuntos reales masivos para la validación de escalabilidad (Sección 4.4).

Nombre	n	m	Clases	Origen
CoverType	581 012	54	2 (clase 1 vs resto)	UCI / sklearn
SUSY	5 000 000	18	2	UCI
HEPMASS	10 500 000	28	2	UCI

4.1.2 Protocolo y métricas de evaluación

Los análisis de calidad emplean una validación cruzada de cinco pliegues estratificada repetida con cinco semillas (42, 123, 456, 789, 1234), con un RandomForestClassifier [43] de cien árboles y profundidad máxima diez como clasificador posterior a la selección [44]. En cada caso se seleccionan las diez variables de mayor relevancia (selección completa en Moons y Circles, que solo tienen dos). La estratificación preserva la distribución de clases en cada pliegue [11].

El capítulo utiliza tres métricas, cada una asociada a un eje del análisis:

- **F1 macro** (calidad de la selección): promedia el F1 de cada clase con igual peso, siendo robusto al desequilibrio. Se agrega sobre pliegues y semillas:

$$\overline{F1} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \frac{1}{K} \sum_{i=1}^{K} F1_{\text{macro}}^{(s,i)} \tag{4.1}$$

donde $\mathcal{S}$ es el conjunto de semillas y $K = 5$ el número de pliegues.

- **Tiempo de ajuste** (eficiencia): tiempo de pared de la llamada a `fit()` del selector. Se mide con una única semilla, ya que el tiempo de ejecución es prácticamente determinista y el objetivo es la tendencia de escala, no la varianza entre semillas.
- **Emisiones de CO_2eq** (sostenibilidad): medidas con codecarbon a partir del consumo de CPU, según se detalla en la Sección 4.5.

De forma complementaria, la comparación de calidad incorpora la similitud entre los *rankings* de relevancia de cada propuesta y los de ReliefF, medida con la correlación de Spearman y el solapamiento del top-10 (Sección 4.6.2).

4.1.3 Plan de experimentación

La Tabla 4.3 resume los análisis del capítulo, la pregunta que responde cada uno y la métrica que emplea. La búsqueda de hiperparámetros (Sección 4.2) precede al resto por fijar las configuraciones usadas en todos los experimentos posteriores.

Tabla 4.3: Mapa de la experimentación del capítulo.

Análisis (sección)	**Pregunta**	**Métrica**
Eficiencia computacional (4.3)	¿Cuánto más rápido que ReliefF?	tiempo de ajuste
Escalabilidad al límite (4.4)	¿Hasta qué tamaño escala donde ReliefF no llega?	tiempo de ajuste, F1
Huella de carbono (4.5)	¿Cuánto se reduce el consumo energético?	CO_2eq
Preservación de la calidad (4.6)	¿Se mantiene la calidad de selección?	F1 macro, tests
Anatomía de la aceleración (4.7)	¿Qué componente aporta la ganancia?	tiempo, F1

4.1.4 Entorno de ejecución

La experimentación combina la implementación propia de las dos variantes con un conjunto de bibliotecas de código abierto ampliamente contrastadas, resumidas en la Tabla 4.4.

Todas las mediciones se realizaron sobre un AMD Ryzen 7 7800X3D (8 núcleos / 16 hilos, 4,2–5,0 GHz), 48 GB DDR5 y Windows 11. Para que las comparaciones de tiempo sean fiables, las ejecuciones se llevaron a cabo sobre la misma máquina y sin otras cargas en paralelo, y las versiones de las dependencias se fijaron de antemano, ya que tanto la búsqueda de vecinos como la medición de tiempo y de emisiones dependen de ellas. El detalle completo del entorno y los pasos para reproducir los experimentos desde el repositorio se recogen en el Anexo A.

Tabla 4.4: Entorno software empleado en la experimentación.

Componente	Descripción
Python 3.11	Lenguaje de implementación
scikit-learn 1.x	Clasificador posterior, métricas y datasets base
hnswlib	Índice HNSW (implementación C++)
Numba	Compilación JIT del kernel de puntuación
joblib	Paralelización por clase en Proto-ReliefF
codecarbon	Medición de emisiones de CO_2
skrebate	Implementación de referencia de ReliefF

4.2 Búsqueda de hiperparámetros

El objetivo de la búsqueda es determinar, para cada algoritmo, la configuración que maximiza la calidad de la selección sujeta a un coste temporal razonable. El criterio de selección es la frontera de Pareto en el espacio F1-tiempo, donde una configuración domina a otra si consigue igual o mayor F1 en igual o menor tiempo. Esta aproximación evita fijar a priori la importancia relativa entre calidad y eficiencia, dejando que la estructura de los datos determine el conjunto de soluciones no dominadas.

4.2.1 Espacio de búsqueda

La Tabla 4.5 describe los espacios de búsqueda. La rejilla de HNSW-ReliefF cubre cuatro parámetros del índice (48 combinaciones) y la de Proto-ReliefF tres (32 combinaciones). Cada configuración se evalúa con el protocolo de la Sección 4.1.2.

Tabla 4.5: Rejillas de búsqueda de hiperparámetros para HNSW-ReliefF y Proto-ReliefF.

Algoritmo	Parámetro	Valores explorados	Configs.
HNSW-ReliefF	`n_neighbors`	$\{5, 10, 15, 20\}$	48
	`M`	$\{8, 16, 32\}$	
	`ef_construction`	$\{100, 200\}$	
	`ef_search`	$\{50, 200\}$	
Proto-ReliefF	`k_protos`	$\{3, 5, 10, 15\}$	32
	`sigma`	$\{0{,}10, 0{,}15, 0{,}20, 0{,}25\}$	
	`use_lvq`	$\{$`True`, `False`$\}$	

4.2.2 Análisis del espacio HNSW-ReliefF

Las Figuras 4.1 y 4.2 muestran los mapas de calor de F1 y de tiempo de ajuste sobre las 48 configuraciones. El efecto de M y `ef_construction` sobre el F1 es prácticamente nulo, ya que dentro del rango explorado el grafo tiene suficiente conectividad para que la búsqueda converja hacia los mismos vecinos con independencia de su valor, coherente con el comportamiento reportado para HNSW en *benchmarks* de recuperación [7]. El parámetro `ef_search` se comporta de forma similar, pues pasar de 50 a 200 no mejora el F1 pero encarece la consulta un 42 %.

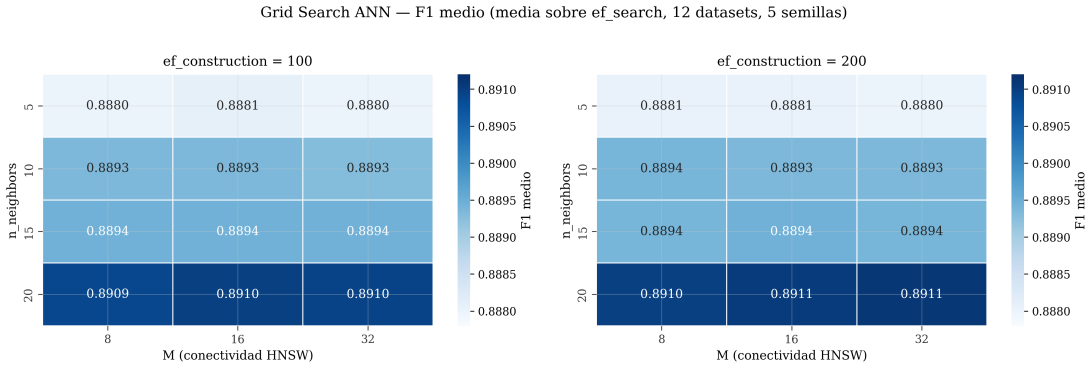


Figura 4.1: Mapa de calor del F1 medio de HNSW-ReliefF sobre las 48 configuraciones del *grid search*.

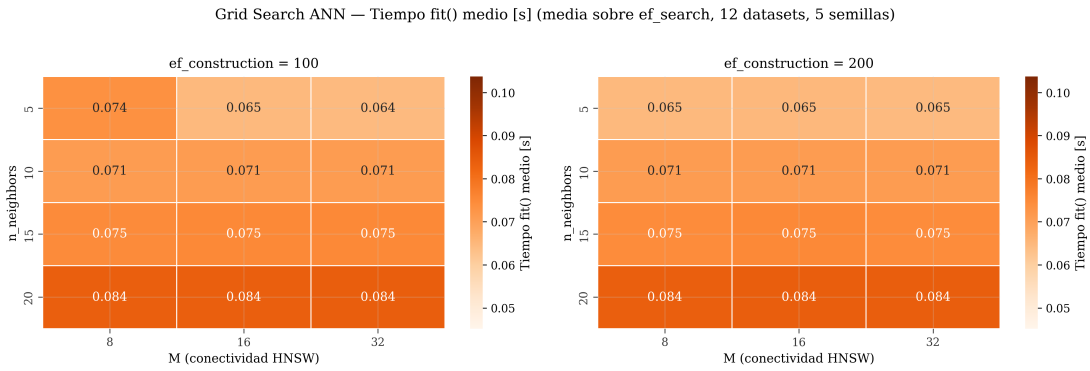


Figura 4.2: Mapa de calor del tiempo de ajuste medio de HNSW-ReliefF sobre las 48 configuraciones del *grid search*.

El único parámetro con influencia clara sobre el F1 es `n_neighbors`, cuyo rendimiento crece de forma monótona de $k = 5$ a $k = 20$ a costa de un mayor tiempo de ajuste. La frontera de Pareto (Figura 4.3) sitúa el codo coste/beneficio en $k = 10$, la configuración `n_neighbors`$= 10$, $M = 8$, `ef_construction`$= 100$, `ef_search`$= 50$ ofrece un F1 a 0,0014 puntos del máximo a menos de la mitad del tiempo.

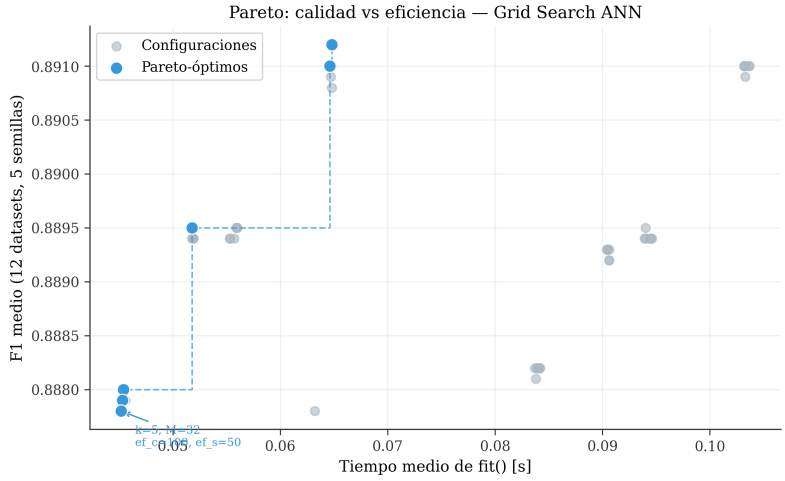


Figura 4.3: Frontera de Pareto F1 vs. tiempo de ajuste para las 48 configuraciones de HNSW-ReliefF. Los puntos dominados se muestran en gris; el punto Pareto-óptimo seleccionado se destaca.

4.2.3 Análisis del espacio Proto-ReliefF

Las Figuras 4.4 y 4.5 comparan los mapas de calor de F1 de Proto-ReliefF sin y con refinamiento LVQ. La diferencia es marginal, ya que activar LVQ aporta 0,0004 puntos de F1 de media a costa de un 20 % más de tiempo. Como se discutió en la Sección 3.3.6, la interacción adversa entre LVQ y el snap-to-grid explica esta relación coste/beneficio desfavorable y confirma la decisión de mantener LVQ desactivado.

Sin LVQ, el parámetro más influyente es `k_protos`, cuyo F1 crece de forma apreciable hasta $k_p = 10$ y marginal después. El parámetro `sigma` influye poco en el F1 pero sí en el tiempo, al controlar el número de prototipos por clase. El mejor compromiso se sitúa en $k_p = 15$, $\sigma = 0{,}15$ (Figura 4.6).

4.2.4 Configuración óptima seleccionada

La Tabla 4.6 resume la configuración elegida para cada algoritmo frente al valor que maximiza el F1 de forma aislada. La selección Pareto-óptima sacrifica entre 0,0004 y 0,0014 puntos de F1 a cambio de reducciones de tiempo de entre el 20 % y el 42 %. Estas configuraciones quedan registradas en el fichero de configuración del proyecto y se usan en el resto del capítulo.

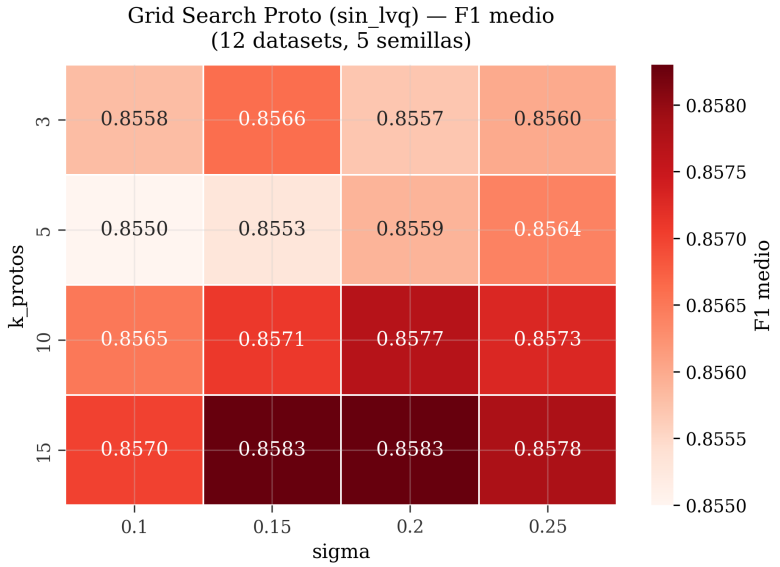


Figura 4.4: Mapa de calor del F1 medio de Proto-ReliefF según `k_protos` (filas) y `sigma` (columnas), sin refinamiento LVQ (`use_lvq=False`).

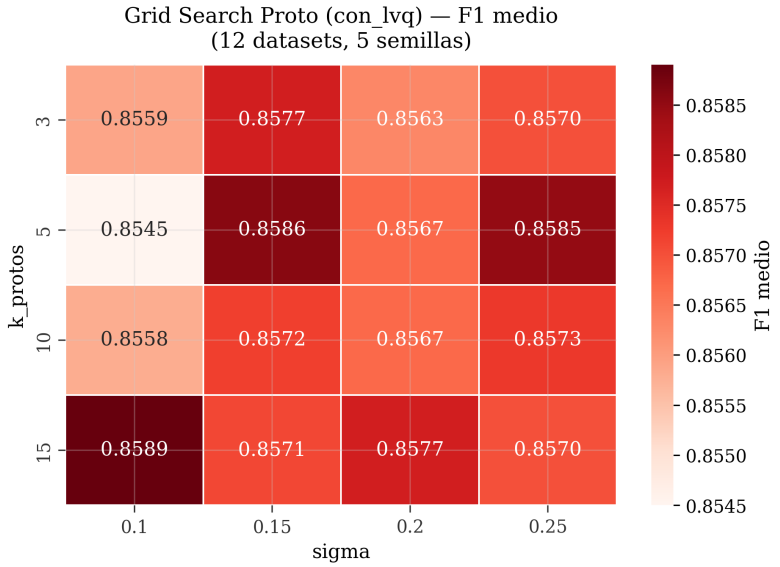


Figura 4.5: Mapa de calor del F1 medio de Proto-ReliefF según `k_protos` (filas) y `sigma` (columnas), con refinamiento LVQ (`use_lvq=True`).

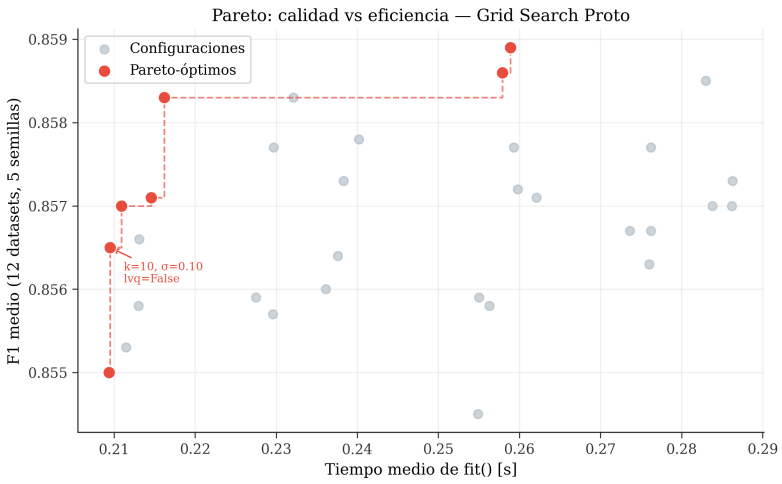


Figura 4.6: Frontera de Pareto F1 vs. tiempo de ajuste para las 32 configuraciones de Proto-ReliefF.

Tabla 4.6: Configuraciones de máximo F1 y Pareto-óptimas seleccionadas para cada algoritmo.

Algoritmo	Parámetro	Máx. F1	Seleccionado	Justificación
HNSW-ReliefF	`n_neighbors`	20	10	Codo coste/beneficio
	`M`	32	8	Efecto en F1: nulo
	`ef_construction`	200	100	Efecto en F1: nulo
	`ef_search`	50	50	Igual F1, $-42\,\%$ tiempo vs. 200
Proto-ReliefF	`k_protos`	15	15	Mejor F1
	`sigma`	0,10	0,15	Mejor ratio F1/tiempo
	`use_lvq`	True	False	$+0{,}0004$ F1 a costa de $+20\,\%$ tiempo

4.3 Eficiencia computacional

La motivación del trabajo es reducir el coste cuadrático de ReliefF. Esta sección cuantifica la ganancia en tiempo de ajuste de las dos variantes en función del tamaño del conjunto, verificando que las predicciones de complejidad del Capítulo 3 se cumplen en la práctica.

4.3.1 Protocolo de medición

Se generan conjuntos sintéticos de tamaño creciente con 30 variables y 20 informativas, y se registra el tiempo de ajuste de los cuatro algoritmos de la familia, ReliefF y MultiSURF (exactos, de la biblioteca de referencia) frente a las dos propuestas. ReliefF y MultiSURF solo se miden hasta $n = 10\,000$, por encima de lo cual su coste cuadrático los hace inviables. Cada punto se mide con una única semilla, dado que el tiempo de ajuste es prácticamente determinista. La primera ejecución de HNSW-ReliefF en la sesión incluye el coste de compilación

JIT de Numba, un *overhead* fijo cuantificado aparte en la Sección 4.7.3; los puntos en estado estacionario son los relevantes para la tendencia de escala.

4.3.2 Resultados

La Tabla 4.7 recoge los tiempos en el rango donde los cuatro algoritmos son ejecutables y la Figura 3.3 muestra el ajuste de la ley de potencia $t \propto n^{\alpha}$ en escala log-log sobre todo el rango medido.

Tabla 4.7: Tiempo de ajuste (s) de los cuatro algoritmos de la familia en el rango donde los exactos son ejecutables. La aceleración es $t_{\text{ReliefF}}/t_{\text{HNSW}}$.

n	ReliefF (s)	MultiSURF (s)	HNSW-ReliefF (s)	Proto-ReliefF (s)	Acel. HNSW
1 000	0,638	6,81	0,056	0,207	11,4×
2 000	1,701	27,25	0,099	0,289	17,1×
5 000	7,069	167,8	0,180	0,575	39,3×
10 000	24,04	673,8	0,353	0,693	68,1×

4.3.3 Análisis

El ajuste log-log confirma la separación cualitativa entre las dos generaciones. Los métodos exactos crecen de forma superlineal, MultiSURF, que no submuestrea, exhibe el exponente cuadrático esperado ($\alpha = 1{,}99$), y ReliefF un exponente intermedio ($\alpha = 1{,}50$) que, aun por debajo de 2 en este rango limitado, basta para volverlo inviable a gran escala. Las dos propuestas, en cambio, crecen de forma cuasi-lineal, HNSW-ReliefF con $\alpha = 0{,}90$ gracias al índice y al kernel Numba, y Proto-ReliefF con $\alpha = 0{,}62$ por su compresión a un número acotado de prototipos. La consecuencia práctica (Tabla 4.7) es una aceleración de HNSW-ReliefF frente a ReliefF que parte de un orden de magnitud y crece hasta $68\times$ a $n = 10^4$, y de varios órdenes de magnitud frente a MultiSURF. En este rango HNSW-ReliefF es más rápido que Proto-ReliefF, pero su menor exponente anticipa que Proto-ReliefF será más rápido en el procesamiento de grandes cantidades de datos, como se analiza en la Sección 4.4.

4.4 Escalabilidad al límite

La sección anterior comparó la familia completa en el rango donde los métodos exactos siguen siendo ejecutables (hasta $n = 10\,000$). Esta sección lleva HNSW-ReliefF y Proto-ReliefF al objetivo que motiva el trabajo, esto es, procesar desde conjuntos de cientos de miles a millones de instancias, donde ReliefF es inviable tanto por tiempo (coste $\mathcal{O}(n^2)$) como por memoria (la matriz de distancias crece con $\mathcal{O}(n^2)$ y desborda la RAM disponible mucho antes del millón de instancias).

4.4.1 Protocolo

Se mide el tiempo de `fit()` de HNSW-ReliefF y Proto-ReliefF sobre conjuntos sintéticos de tamaño creciente hasta $n = 10^7$ (30 variables, 20 informativas), con una única semilla. Adicionalmente se evalúan los tres conjuntos reales masivos de la Tabla 4.2 (CoverType, SUSY y HEPMASS), midiendo tanto el tiempo de ejecución del método de selección como el F1 macro de un Random Forest entrenado con las variables seleccionadas. Dado el tamaño de estos conjuntos de datos, se emplea una única partición entrenamiento-prueba, ya que la validación cruzada resulta computacionalmente inviable. Las variables se estandarizan antes de la selección, paso imprescindible al operar con distancias euclídeas sobre datos reales de escalas heterogéneas. ReliefF se omite por inviabilidad debido a su elevado coste computacional. Su límite práctico de aplicación se estima a partir de la extrapolación de los resultados obtenidos en la Sección 4.3 y del límite de memoria teórico.

4.4.2 Resultados

La Tabla 4.8 recoge los tiempos de ajuste sobre los conjuntos sintéticos en el régimen masivo, donde ReliefF ya no aparece por inejecutable. El dato más relevante es la inversión respecto al rango medio, ya que a partir de aproximadamente $n = 5 \times 10^5$, Proto-ReliefF pasa a ser más rápido que HNSW-ReliefF, y la ventaja se amplía con el tamaño hasta un factor de $7{,}5\times$ a diez millones de instancias.

Tabla 4.8: Tiempo de ajuste (s) en el régimen masivo (una semilla). ReliefF se omite por inviabilidad. El factor compara ambas propuestas entre sí.

n	HNSW-ReliefF (s)	Proto-ReliefF (s)	Proto vs. HNSW
100 000	1,91	2,02	$0{,}9\times$
500 000	17,78	7,95	$2{,}2\times$
1 000 000	55,61	15,48	$3{,}6\times$
5 000 000	522,9	78,35	$6{,}7\times$
10 000 000	1287,6	172,5	$7{,}5\times$

Sobre los conjuntos reales masivos, la Figura 4.7 y la Tabla 4.9 muestran que ambas propuestas completan la selección en tiempos manejables (de segundos a pocos minutos) y producen selecciones con información útil. Proto-ReliefF es, de nuevo, el más rápido en los tres conjuntos analizados. En cuanto a la calidad de la selección, HNSW-ReliefF obtiene mejor F1 en CoverType y SUSY, mientras que Proto-ReliefF lo supera en HEPMASS.

Tabla 4.9: Tiempo de ajuste y F1 macro (*holdout*) de las dos propuestas sobre los conjuntos reales masivos. ReliefF es inejecutable a estas escalas.

Dataset	n	m	HNSW t (s)	HNSW F1	Proto t (s)	Proto F1
CoverType	581 012	54	10,49	0,811	8,03	0,732
SUSY	5 000 000	18	111,1	0,775	53,9	0,736
HEPMASS	10 500 000	28	380,6	0,814	132,8	0,840

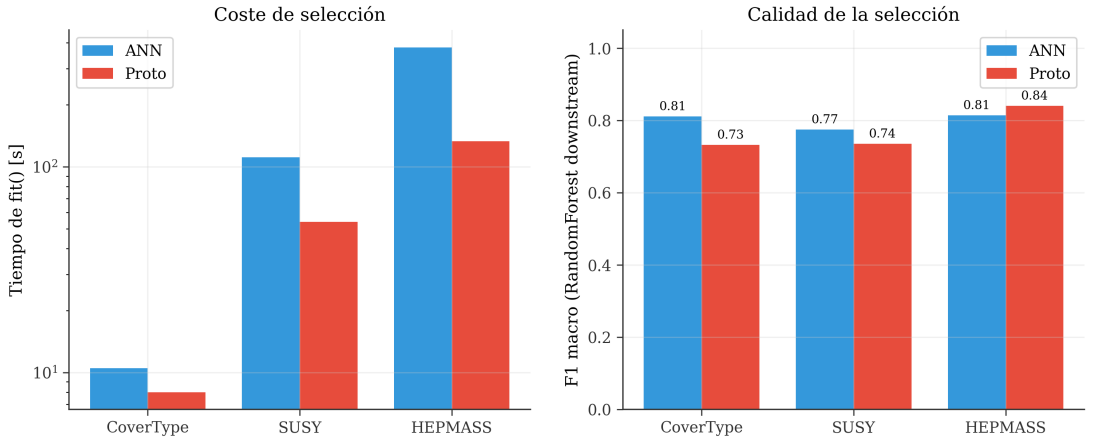


Figura 4.7: Coste y calidad de la selección sobre los conjuntos reales masivos. Izquierda: tiempo de ajuste (escala logarítmica); Proto-ReliefF es más rápido en los tres. Derecha: F1 macro del clasificador posterior; ambas propuestas se mantienen en el rango 0,73–0,84.

4.4.3 Análisis

El resultado central es que ambas propuestas operan con normalidad en un régimen donde ReliefF resulta inviable. Extrapolando la curva de la Sección 4.3, que mide 24 s a $n = 10^4$, el coste cuadrático de ReliefF a diez millones de instancias sería del orden de 10^6 veces mayor, es decir, semanas de cómputo; y antes de llegar a ese tiempo el algoritmo fracasaría por memoria, ya que la matriz de distancias de tamaño $\mathcal{O}(n^2)$ requeriría del orden de 10^{14} valores, muy por encima de cualquier RAM disponible. Frente a ello, HNSW-ReliefF resuelve diez millones de instancias en unos 21 minutos y Proto-ReliefF en menos de tres.

La inversión de rendimiento entre las dos propuestas confirma empíricamente la guía de uso del Capítulo 3. En el rango medio es preferible HNSW-ReliefF, mientras que en el régimen masivo el menor exponente de Proto-ReliefF se impone, ya que su compresión a un número acotado de prototipos por clase hace que el coste de la puntuación deje de depender del tamaño original y que el algoritmo escale de forma esencialmente lineal con una constante muy baja. HNSW-ReliefF, en cambio, debe construir y consultar un índice sobre las n instancias completas, cuyo coste, aunque sublineal por consulta, crece de forma más marcada en

conjunto. Esto sitúa a Proto-ReliefF como la opción preferente cuando el tamaño es extremo, reforzado además por su menor huella de memoria, $\mathcal{O}(p \cdot m)$ frente a $\mathcal{O}(n \cdot M)$ del índice.

Por último, la calidad de la selección se mantiene en un rango útil (F1 entre 0,73 y 0,84) sobre datos reales masivos y heterogéneos, lo que indica que la ventaja en eficiencia no se obtiene a costa de producir selecciones inservibles. El reparto de la mejor calidad entre ambas propuestas según el conjunto es coherente con la equivalencia estadística establecida en la Sección 4.6. Ninguna destaca sistemáticamente sobre la otra en calidad, y la elección entre ellas a gran escala se decide por el coste, donde Proto-ReliefF es claramente superior.

4.5 Huella de carbono

La reducción de cómputo demostrada en las Secciones 4.3 y 4.4 tiene una consecuencia directa, y es que menos tiempo de CPU implica menos energía consumida y, por tanto, menos emisiones. Esta sección cuantifica ese efecto, enmarcado en el objetivo de la Cátedra Inditex-UDC de IA en Algoritmos Verdes de desarrollar algoritmos de bajo impacto energético.

4.5.1 Metodología de medición

Las emisiones de CO_2 equivalente se miden instrumentando cada llamada a `fit()` con codecarbon [6], que las estima a partir del consumo de CPU medido por RAPL y la intensidad de carbono de la red eléctrica local. Los resultados se expresan en microgramos de CO_2eq por ejecución. Adicionalmente se calcula el **ratio de eficiencia verde** (F1 por gramo de CO_2eq), que mide cuánta calidad de selección se obtiene por unidad de emisión.

4.5.2 Resultados y análisis

La Tabla 4.10 recoge las emisiones para una selección de conjuntos y la Figura 4.8 sitúa a los tres algoritmos en el espacio calidad–emisiones.

HNSW-ReliefF emite de media un 41 % menos que ReliefF con un F1 equivalente, lo que lo convierte en la opción dominante de la frontera de Pareto (Figura 4.8). El ahorro es mayor cuanto mayor es el conjunto (Tabla 4.10), coherente con la diferencia de complejidad, solo en los conjuntos más pequeños el coste de construir el índice supera al ahorro del bucle. Proto-ReliefF reduce las emisiones un 35 % y ocupa una posición intermedia.

El ratio de eficiencia verde (F1 obtenido por gramo de CO_2eq) resume esta ventaja en una sola magnitud (Figura 4.9), donde HNSW-ReliefF supera a ReliefF en un 39 %, obteniendo más calidad de selección por cada unidad de emisión. En el marco de la IA Verde [4], un ahorro de esta magnitud sin coste en calidad es especialmente relevante cuando la selección forma parte de *pipelines* repetidos sobre muchas ejecuciones.

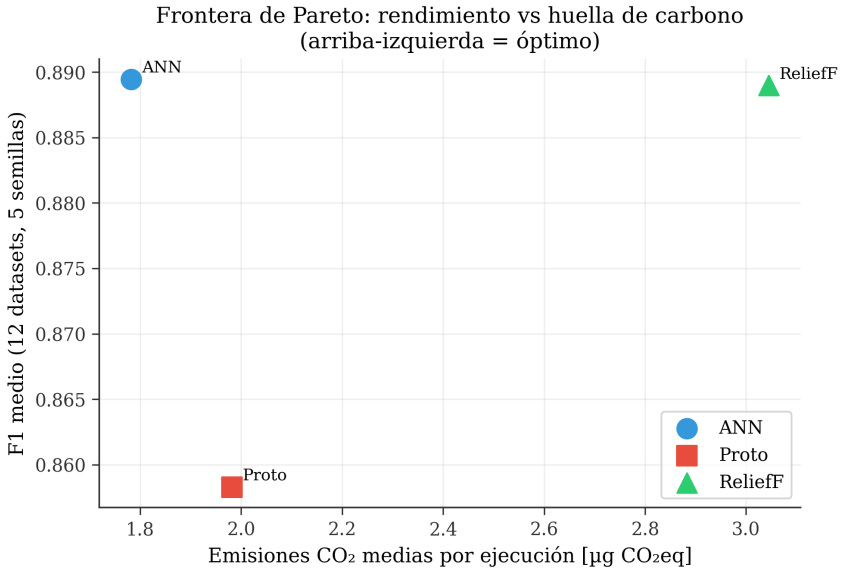


Figura 4.8: Frontera de Pareto F1 vs. emisiones de CO_2eq (media sobre los doce conjuntos). HNSW-ReliefF supera a ReliefF, F1 equivalente con un 41 % menos de emisiones.

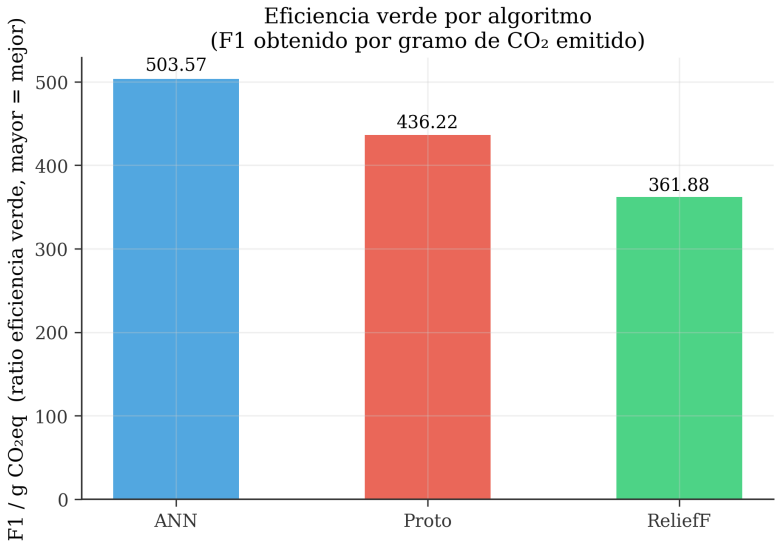


Figura 4.9: Ratio de eficiencia verde (F1 conseguido por gramo de CO_2eq) de cada algoritmo. Un valor más alto indica más calidad de selección por unidad de emisión.

Tabla 4.10: Emisiones medias de CO_2eq por ejecución (μg) para una selección representativa de conjuntos. La columna "Reducción" indica el factor de ahorro de HNSW-ReliefF frente a ReliefF.

Dataset	ReliefF (μg)	HNSW-ReliefF (μg)	Proto-ReliefF (μg)	Reducción
Breast_Cancer	2,060	2,465	2,560	0,84×
Digits	5,011	1,760	1,992	2,85×
Corral	2,131	1,678	1,837	1,27×
CorrAL100	4,529	1,829	2,062	2,48×
AltaDim	2,749	1,659	1,870	1,66×
Grande	7,985	1,909	2,378	4,18×
Media	**3,044**	**1,782**	**1,982**	**1,71×**

4.6 Preservación de la calidad de selección

Las secciones anteriores establecieron que HNSW-ReliefF y Proto-ReliefF son sustancialmente más rápidos, escalables y eficientes en emisiones. Esta sección comprueba el contrapunto necesario, que esa ganancia no se obtiene a costa de la calidad de la selección. Solo tiene sentido sustituir a ReliefF por un método más eficiente si produce selecciones de calidad comparable.

4.6.1 Resultados por conjunto de datos

La Tabla 4.11 recoge el F1 macro medio sobre las 25 evaluaciones por celda. HNSW-ReliefF iguala a ReliefF en prácticamente todos los conjuntos (su media global, 0,8894, es indistinguible de la de ReliefF, 0,8890). Proto-ReliefF es competitivo en general, con una media 3,1 puntos de F1 inferior atribuible casi por completo al caso Circles, donde la compresión por clustering de una frontera circular concéntrica degrada la selección, de acuerdo con la limitación de *clustering* multimodal descrita en la Sección 3.4.2. En CorrAL100 los tres algoritmos recuperan idénticamente las cuatro variables relevantes (F1 perfecto), lo que confirma que las variantes preservan la capacidad de ReliefF para detectar interacciones entre variables, su principal propiedad.

La equivalencia en CorrAL100 puede examinarse con más detalle observando la posición media de cada variable en el ranking, ya que este conjunto tiene *ground truth* conocido. La Figura 4.10 muestra ese comportamiento. Los tres algoritmos sitúan las cuatro variables causales (f_0–f_3) en las posiciones más altas, relegan las variables de ruido al fondo del ranking y colocan la variable trampa (f_5, correlada con la clase pero causalmente irrelevante) en lo más alto. Lo relevante es que las tres distribuciones de posiciones son prácticamente idénticas, la aproximación de HNSW-ReliefF y la compresión de Proto-ReliefF no alteran qué variables se

Tabla 4.11: F1 macro medio ± desviación típica (5 pliegues × 5 semillas). En negrita, el mejor resultado por fila.

Dataset	ReliefF	HNSW-ReliefF	Proto-ReliefF
Breast_Cancer	**0,9705 ± 0,0018**	0,9696 ± 0,0025	0,9503 ± 0,0037
Wine	**0,9867 ± 0,0035**	0,9852 ± 0,0036	0,9835 ± 0,0041
Digits	**0,9338 ± 0,0017**	0,9173 ± 0,0032	0,8886 ± 0,0052
Corral	0,7911 ± 0,0054	**0,7999 ± 0,0118**	0,7869 ± 0,0186
CorrAL100	**1,0000 ± 0,0000**	**1,0000 ± 0,0000**	**1,0000 ± 0,0000**
XOR	0,9522 ± 0,0073	**0,9555 ± 0,0091**	0,9152 ± 0,0079
Moons	0,9118 ± 0,0014	**0,9119 ± 0,0014**	0,8887 ± 0,0050
Circles	**0,8720 ± 0,0046**	**0,8720 ± 0,0046**	0,5880 ± 0,0061
AltaDim	0,8852 ± 0,0040	0,8884 ± 0,0064	**0,8990 ± 0,0072**
Desbalanceado	0,6969 ± 0,0128	0,6959 ± 0,0161	**0,7202 ± 0,0103**
Ruidoso	0,8067 ± 0,0063	0,8118 ± 0,0025	**0,8201 ± 0,0096**
Grande	0,8607 ± 0,0028	**0,8659 ± 0,0040**	0,8589 ± 0,0040
Media	0,8890	**0,8894**	0,8583

consideran relevantes ni en qué orden, lo que confirma que ambas preservan la capacidad de detección de interacciones de ReliefF. La susceptibilidad común a la trampa f_5 no es un defecto de las variantes sino una propiedad heredada del criterio de relevancia de ReliefF, que no distingue correlación de causalidad.

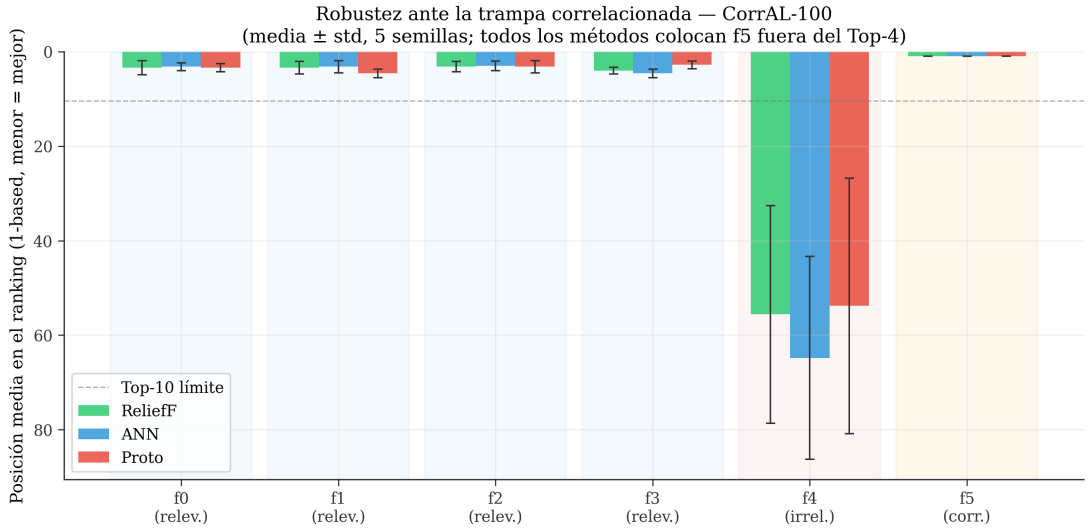


Figura 4.10: Posición media en el ranking de las variables clave de CorrAL100 (1 = más relevante) para los tres algoritmos, sobre cinco semillas. Las cuatro variables causales f_0–f_3 y la trampa f_5 ocupan posiciones equivalentes en los tres métodos; el ruido f_4 se descarta al fondo.

4.6.2 Similitud de los rankings

La comparación anterior se apoya en el F1 del clasificador entrenado sobre las variables seleccionadas, una medida indirecta de la calidad. Para comprobar de forma directa que las propuestas no solo conducen a un F1 parecido, sino que emiten el mismo juicio de relevancia que ReliefF, esta sección compara los rankings entre sí mediante dos métricas, las cuales son la correlación de Spearman entre los vectores de pesos completos, que mide si se conserva el orden de relevancia, y el solapamiento del top-10, que mide qué fracción de las variables seleccionadas coinciden con las de ReliefF. Ambas se promedian sobre cinco semillas y se calculan sobre los conjuntos de tamaño pequeño y medio con suficientes variables; CorrAL100 se analiza por separado con su *ground truth* en la Figura 4.10.

Los resultados (Tabla 4.12 y Figura 4.11) muestran que HNSW-ReliefF reproduce el ranking de ReliefF casi exactamente. Una correlación de Spearman media de 0,92 y un solapamiento medio del top-10 de 0,88 indican que la búsqueda aproximada apenas altera el orden de relevancia ni qué variables se eligen. Proto-ReliefF conserva también lo esencial, con una correlación media de 0,80 y un solapamiento de 0,78, algo por debajo de HNSW-ReliefF, lo que es coherente con que su compresión introduce más perturbación en los pesos y con su F1 medio ligeramente inferior (Sección 4.6.1).

Conviene leer las dos métricas en conjunto. La correlación de Spearman se calcula sobre todas las variables, incluidas las irrelevantes, cuyos pesos son próximos a cero y se ordenan de

Tabla 4.12: Similitud del ranking de relevancia de las propuestas frente a ReliefF: correlación de Spearman (ρ) entre los pesos y solapamiento del top-10 (fracción de las 10 variables seleccionadas en común), promediados sobre cinco semillas.

Dataset	**ρ HNSW**	**ρ Proto**	**Top-10 HNSW**	**Top-10 Proto**
Breast_Cancer	0,95	0,76	1,00	0,80
Wine	0,94	0,64	0,90	0,80
Digits	0,96	0,90	0,80	0,60
Corral	0,74	0,66	0,76	0,76
AltaDim	0,89	0,80	0,86	0,72
Desbalanceado	0,93	0,84	0,90	0,80
Ruidoso	0,94	0,89	0,92	0,90
Grande	0,97	0,95	0,90	0,82
Media	**0,92**	**0,80**	**0,88**	**0,78**

forma casi arbitraria, lo que reduce la correlación aunque las variables importantes coincidan. Por eso, en los conjuntos donde la correlación baja (Wine, Corral), el solapamiento del top-10 se mantiene alto. La discrepancia se concentra en las características de las últimas posiciones del ranking, menos relevantes o donde existen otras con capacidad predictiva similar, sin afectar demasiado a la selección. La conclusión es que ambas propuestas preservan el ranking de ReliefF, y no solo el rendimiento del clasificador posterior; HNSW-ReliefF de forma casi exacta y Proto-ReliefF en lo sustancial.

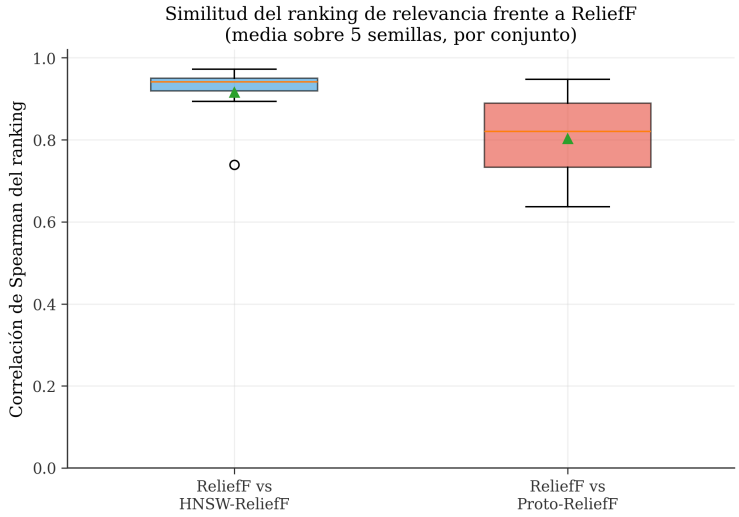


Figura 4.11: Distribución de la correlación de Spearman entre el ranking de ReliefF y el de cada propuesta. Cada caja resume los ocho valores, uno por conjunto de datos: la línea central es la mediana, la caja abarca el rango intercuartílico (el 50 % central de los conjuntos), los bigotes llegan al mínimo y al máximo, y el triángulo marca la media. Cuanto más alta y compacta es la caja, más fielmente y de forma más consistente se preserva el ranking. HNSW-ReliefF queda más arriba y más concentrado que Proto-ReliefF, aunque ambos mantienen correlaciones elevadas.

4.6.3 Robustez frente al ruido

Un último aspecto de la calidad es la robustez, ¿se degradan las propuestas más rápido que ReliefF cuando los datos se corrompen? Conviene precisar la pregunta, ya que todos los selectores empeoran con el ruido, también ReliefF; lo relevante no es la degradación absoluta, sino si la aproximación de HNSW-ReliefF o la compresión de Proto-ReliefF añaden fragilidad respecto al método exacto.

Para medirlo se emplea CorrAL100, que tiene *ground truth* conocido, cuatro variables causales, una variable correlada con la clase pero causalmente irrelevante y noventa y cinco de ruido. Se inyecta una fracción creciente de ruido en las etiquetas, hasta el 25 %, y, sobre los mismos datos corrompidos, se ajustan los tres algoritmos y se mide la recuperación, es decir, cuántas de las cuatro variables relevantes quedan entre las diez seleccionadas, promediada sobre cinco semillas.

La Figura 4.12 muestra el resultado. Hasta un 20 % de etiquetas corruptas, los tres algoritmos recuperan las cuatro variables relevantes de forma idéntica. HNSW-ReliefF reproduce exactamente la recuperación de ReliefF en todo el rango de ruido, mientras que Proto-ReliefF solo se separa de forma marginal en el nivel más alto, con una recuperación de 0,95 frente a 1,0 al 25 %. La conclusión es que ni la búsqueda aproximada ni la compresión incrementan

la sensibilidad al ruido, ambas propuestas heredan la robustez de ReliefF en lugar de añadir fragilidad propia.

Este conjunto sirve además para la parte de redundancia, pues incluye una variable correlada con la clase pero causalmente irrelevante. Como se observó en la Figura 4.10, los tres algoritmos la sitúan en lo alto del ranking por igual, la susceptibilidad a esa correlación espuria es una propiedad heredada del criterio de ReliefF, no un defecto introducido por las propuestas, lo que confirma que su comportamiento frente a la redundancia es también equivalente al del método original.

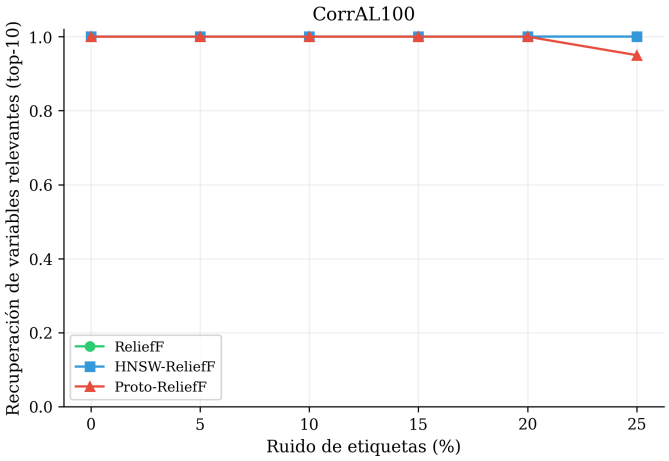


Figura 4.12: Recuperación de las cuatro variables relevantes de CorrAL100 (entre las diez seleccionadas) en función del ruido de etiquetas, promediada sobre cinco semillas. Las tres curvas se solapan hasta el 20 %; HNSW-ReliefF sigue a ReliefF en todo el rango y Proto-ReliefF solo cae de forma marginal al 25 %.

4.6.4 Análisis estadístico

Para determinar si las diferencias son significativas se aplican tres procedimientos no paramétricos. El test de Wilcoxon *signed-rank* pareado compara dos algoritmos sobre los doce conjuntos (que actúan como bloques pareados) sin asumir normalidad, procedimiento estándar para comparar algoritmos sobre múltiples *datasets* [45]. El test de Friedman evalúa si existe un ordenamiento global consistente entre los tres. El d de Cohen cuantifica el tamaño del efecto [46], con corrección de Bonferroni para las tres comparaciones pareadas. La Figura 4.13 ilustra la distribución de partida y la Tabla 4.13 recoge los resultados.

Ninguna comparación alcanza significación estadística, ni con $\alpha = 0{,}05$ ni tras la corrección de Bonferroni, y los tamaños de efecto son triviales o pequeños ($|d| < 0{,}31$). El test de Friedman no detecta un ordenamiento global consistente, la relación de dominancia depende del *dataset* y no es estable. La conclusión es que ambas variantes son estadísticamente equiva-

Tabla 4.13: Tests estadísticos de comparación entre algoritmos ($\alpha = 0{,}05$, corrección de Bonferroni). "No aplica" indica que el estadístico no está definido para el test de Friedman.

Comparación	Test	Estadístico	p-valor	Sig.	d Cohen	p Bonf.	Sig. Bonf.
ReliefF vs HNSW-ReliefF	Wilcoxon	$W = 25{,}0$	0,5195	No	$-0{,}005$	1,000	No
ReliefF vs Proto-ReliefF	Wilcoxon	$W = 17{,}0$	0,1748	No	$+0{,}295$	0,524	No
HNSW-ReliefF vs Proto-ReliefF	Wilcoxon	$W = 15{,}0$	0,1230	No	$+0{,}301$	0,369	No
ReliefF / HNSW / Proto	Friedman	$\chi^2 = 3{,}45$	0,1778	No	No aplica	0,533	No

lentes a ReliefF en calidad de selección, lo que valida el prerrequisito sobre el que se sostiene todo el argumento de eficiencia del capítulo, la ganancia en velocidad, escala y emisiones no se paga con pérdida de calidad.

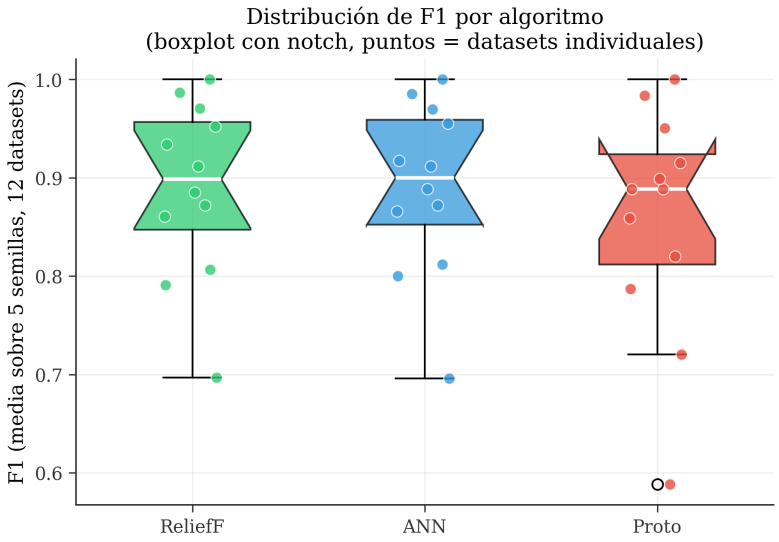


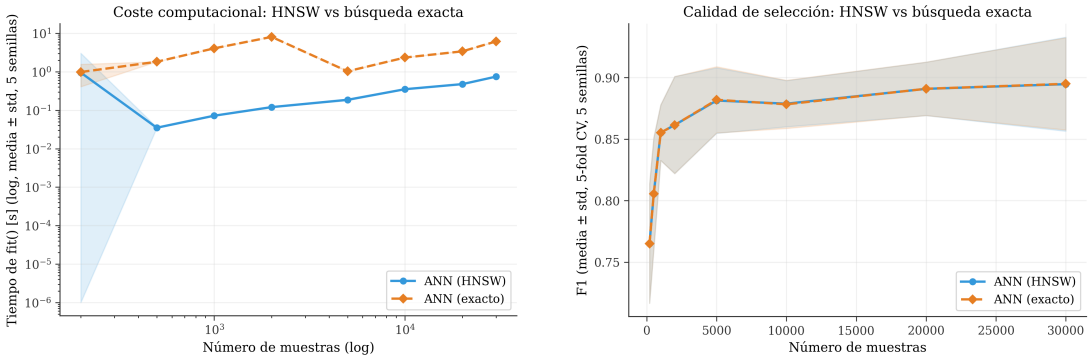
Figura 4.13: Distribución del F1 macro por algoritmo sobre los doce conjuntos. La línea central es la mediana; los bigotes representan $1{,}5 \times$ IQR; los puntos son los valores por *dataset*.

4.7 Anatomía de la aceleración

Las secciones anteriores trataron HNSW-ReliefF como un sistema completo. Esta sección lo descompone para identificar de dónde procede la aceleración, mediante tres análisis, los cuales son la comparación directa entre el índice HNSW y la búsqueda exacta dentro del mismo *pipeline*, un estudio de ablación incremental, y la cuantificación del coste de calentamiento del kernel JIT.

4.7.1 Índice HNSW frente a búsqueda exacta

Este experimento aísla el efecto del índice comparando dos variantes idénticas del *pipeline* que solo difieren en el mecanismo de búsqueda (HNSW aproximado frente a exacto), sobre conjuntos de 200 a 100 000 instancias. La Figura 4.14 muestra ambas dimensiones, el F1 de las dos variantes es indistinguible en todo el rango (lo que confirma a nivel de componente la equivalencia de calidad de la Sección 4.6), mientras que la variante HNSW es más rápida en todo el rango, sin punto de cruce a favor de la búsqueda exacta. La coincidencia exacta del F1 en los conjuntos más pequeños no se debe a que el índice se desactive (HNSW se emplea en todo el rango), sino a que con pocas instancias el grafo recupera los vecinos exactos, con un *recall* cercano al 100 %, la aproximación produce entonces la misma selección que la búsqueda exacta. Las diferencias, siempre mínimas y restringidas a la cuarta cifra decimal, solo afloran al crecer el tamaño y reducirse ligeramente el *recall* del índice.



(a) Tiempo de ajuste en función de n.

(b) F1 macro en función de n.

Figura 4.14: Variante con índice HNSW frente a búsqueda exacta dentro del mismo *pipeline*, la aproximación acelera sin degradar el F1.

4.7.2 Ablación de componentes

La ablación activa de forma incremental los componentes de HNSW-ReliefF, **ReliefF** de referencia, **ANN-exacto** (kernel Numba con búsqueda exacta), **ANN-HNSW** (añade el índice) y **ANN-completo** (añade el submuestreo adaptativo de `iter_ratio`). La Tabla 4.14 recoge los tiempos y la Figura 4.15 la aceleración acumulada.

La descomposición revela contribuciones de naturaleza distinta. El *kernel* Numba con búsqueda exacta solo compensa a partir de $n \approx 5\,000$, por debajo del cual el coste de las consultas exactas supera el ahorro del *kernel* compilado. El índice HNSW es el componente de mayor impacto individual, reduciendo el tiempo en un orden de magnitud adicional. El submuestreo `iter_ratio` aporta una ganancia creciente con n (nula en conjuntos pequeños, donde el ratio es 1,0). Lo decisivo es que el F1 permanece en 1,000 para las cuatro variantes, lo que

Tabla 4.14: Tiempo de ajuste mediano (s) por variante de la ablación. El F1 es 1,000 para las cuatro variantes en todas las semillas.

n	ReliefF (s)	ANN-exacto (s)	ANN-HNSW (s)	ANN-completo (s)	Aceleración
500	0,318	1,833	0,035	0,035	9,1×
2 000	1,699	10,131	0,137	0,125	13,6×
10 000	23,814	4,612	0,636	0,370	64,4×
30 000	187,773	20,606	2,036	0,750	250,4×

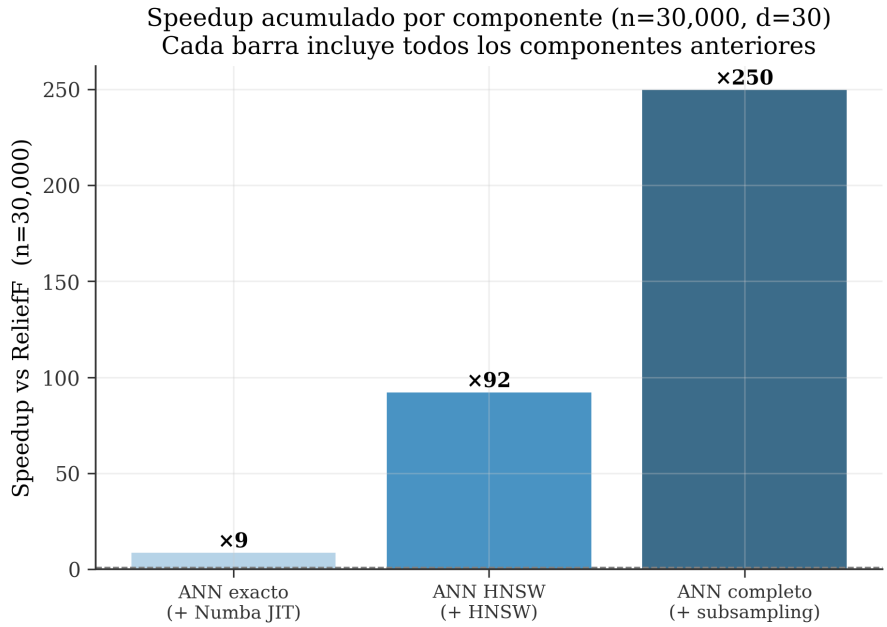


Figura 4.15: Aceleración acumulada de cada variante respecto a ReliefF. El índice HNSW es el factor dominante; el submuestreo adaptativo aporta una ganancia adicional que crece con n.

indica que ninguno de los tres componentes degrada la calidad, de modo que la aceleración total de 250× a $n = 30\,000$ se obtiene sin coste, con contribuciones multiplicativas porque cada componente ataca una fase distinta del algoritmo.

4.7.3 Coste de calentamiento del kernel Numba

El último análisis cuantifica el *overhead* de compilación JIT mencionado en las Secciones 4.3 y 3.2.5. Se ejecutan cinco llamadas consecutivas a `fit()` en una sesión limpia, registrando la primera (*cold*, con compilación) y la mediana de las siguientes (*warm*); ReliefF, que no usa JIT, actúa como control. La Tabla 4.15 resume los resultados.

Los datos confirman dos propiedades. El *overhead* absoluto es constante (entre 4,6 y 4,7 s con independencia de n), coherente con su origen en la compilación del kernel, que depende

Tabla 4.15: Coste de calentamiento del kernel Numba: primera llamada (cold) frente a mediana de las posteriores (warm), con ReliefF como control sin JIT.

n	ANN cold (s)	ANN warm (s)	ReliefF cold (s)	ReliefF warm (s)	Overhead (s)	Factor
500	4,715	0,036	0,317	0,270	4,679	130,9×
2 000	4,730	0,125	1,802	1,762	4,605	37,9×
10 000	5,008	0,372	24,104	24,292	4,636	13,5×

de la firma de tipos y no del volumen de datos. Y el control con ReliefF no muestra diferencia entre la primera llamada y las siguientes, lo que descarta efectos de caché del sistema y atribuye el *overhead* íntegramente al JIT. La implicación es que en cualquier escenario realista (validación cruzada, búsqueda de hiperparámetros, *pipelines* repetidos) el coste se paga una sola vez por sesión y queda amortizado desde la segunda llamada; en la validación cruzada de 25 ejecuciones del capítulo representa menos del 2 % del tiempo total. Para eliminarlo por completo existen la compilación anticipada de Numba o el almacenamiento en caché del *kernel* en disco, alternativas no exploradas en este trabajo.

Capítulo 5

Conclusiones

El punto de partida de este trabajo fue una tensión bien delimitada. ReliefF es uno de los filtros de selección de características más valiosos por su capacidad para detectar interacciones entre variables, una propiedad que los criterios univariados no poseen (Capítulo 2). Sin embargo, su coste cuadrático $\mathcal{O}(n^2 \cdot m)$ lo vuelve inaplicable a los volúmenes de datos actuales, con un impacto que no es solo temporal, sino también energético (Capítulo 1). Sobre esa tensión se articularon dos propuestas algorítmicas, HNSW-ReliefF y Proto-ReliefF (Capítulo 3), y una batería experimental que las evalúa en eficiencia, escalabilidad, emisiones y calidad de selección (Capítulo 4). Este capítulo recoge las conclusiones de ese recorrido sintetizando el trabajo realizado, valorando el cumplimiento de los objetivos planteados, exponiendo las aportaciones, discutiendo los hallazgos principales, reconociendo las limitaciones del enfoque y señalando las líneas de continuación, antes de cerrar con una valoración global de lo conseguido.

5.1 Síntesis del trabajo realizado

El trabajo parte de un diagnóstico y propone una solución de doble vía. El diagnóstico, desarrollado en el estado del arte, sitúa el cuello de botella de toda la familia Relief en un único punto. La búsqueda de los vecinos más cercanos de cada instancia, en su forma exacta, obliga a comparar cada muestra contra todas las demás. Las variantes clásicas de la familia mejoran la calidad de la selección o la robustez al ruido, pero ninguna alivia ese mecanismo de búsqueda, que permanece como la principal barrera de aplicabilidad a gran escala.

La propuesta de este TFG se apoya en una observación que vertebra ambas variantes. El criterio de actualización de pesos de ReliefF es sólido y no necesita modificación, de modo que la eficiencia debe ganarse cambiando *cómo* se obtienen los vecinos, no *qué* se hace con ellos. De esa idea surgen dos estrategias que renuncian a garantías distintas para atacar el mismo problema desde ángulos complementarios. HNSW-ReliefF renuncia a la exactitud de la bús-

queda y la sustituye por consultas sobre un índice de grafo navegable jerárquico, reduciendo el coste por consulta de lineal a logarítmico. Proto-ReliefF renuncia al tamaño del conjunto de búsqueda y comprime las instancias en un número acotado de prototipos representativos antes de aplicar el algoritmo exacto sobre ellos. En ambos casos, la función de relevancia permanece intacta, lo que garantiza que la capacidad de detectar interacciones se conserva.

Esa decisión de diseño se tradujo en dos implementaciones cuidadas, descritas y justificadas en el Capítulo 3, y en un protocolo experimental que las contrasta no solo entre sí, sino frente al ReliefF original y a una variante exacta más costosa (MultiSURF). El Capítulo 4 recorre ese protocolo de forma incremental. Primero fija las configuraciones mediante una búsqueda de hiperparámetros, después cuantifica la ganancia en tiempo, la lleva al límite de la escala masiva, traduce el ahorro de cómputo a emisiones, comprueba que la calidad de la selección no se degrada y, por último, descompone la aceleración para atribuirla a sus componentes. El resultado es una evaluación que sostiene cada afirmación de eficiencia sobre la verificación previa de que la calidad se mantiene.

La inclusión de MultiSURF en la comparación de eficiencia no es accesoria, sino que cumple la función de control, ya que al ser una variante exacta que no submuestrea exhibe el crecimiento cuadrático puro y sirve de referencia del coste que se trata de evitar. Frente a ese control, la separación entre las dos generaciones de métodos (los exactos, que crecen de forma superlineal, y las propuestas, que lo hacen de forma cuasi-lineal o sublineal) se observa de manera nítida y no como una mejora marginal. Este contraste, junto con la elección de conjuntos con *ground truth* conocido para verificar qué variables se seleccionan, ilustra el criterio que ha guiado el diseño experimental, en el que cada medición se acompaña de la referencia que permite interpretarla.

5.2 Grado de cumplimiento de los objetivos

El objetivo principal del trabajo era diseñar, implementar y evaluar versiones sostenibles de ReliefF que conservasen una calidad de selección similar a la del método original reduciendo de forma significativa su coste computacional y energético (Sección 1.1). Ese objetivo se considera cumplido. Las dos variantes propuestas son estadísticamente equivalentes a ReliefF en calidad de selección (Sección 4.6.4) y, al mismo tiempo, reducen el tiempo de ejecución en más de dos órdenes de magnitud (Sección 4.3) y las emisiones en más de un tercio (Sección 4.5). El cumplimiento del objetivo general se sostiene sobre el de los objetivos específicos, que se valoran a continuación.

- *Comprender ReliefF y sus limitaciones.* El Capítulo 2 reconstruye la familia Relief desde el algoritmo original hasta sus variantes modernas y formaliza el origen del coste cuadrático en la búsqueda de vecinos, identificándolo como la barrera que el resto del

trabajo aborda. Este análisis fundamenta todas las decisiones posteriores.

- *Diseñar e implementar la variante basada en búsqueda aproximada.* HNSW-ReliefF, descrito en la Sección 3.2, integra un índice HNSW por clase en el esquema de puntuación de ReliefF. Su validación empírica (Sección 4.3) confirma una aceleración que crece con el tamaño del conjunto hasta $68\times$ a $n = 10^4$, con una calidad indistinguible de la del método original.

- *Diseñar e implementar la variante basada en prototipos.* Proto-ReliefF, descrito en la Sección 3.3, sustituye las instancias originales por un conjunto reducido de representantes obtenidos mediante *clustering* supervisado. Su evaluación en el régimen masivo (Sección 4.4) demuestra que resuelve problemas de diez millones de instancias en menos de tres minutos, allí donde ReliefF es directamente inejecutable.

- *Estudiar la robustez frente al ruido.* La robustez se evaluó con un experimento dedicado (Sección 4.6.3) que inyecta niveles crecientes de ruido en las etiquetas y mide cuántas variables relevantes conserva cada algoritmo frente a ReliefF. El resultado confirma que ni la búsqueda aproximada ni la compresión añaden fragilidad, ya que HNSW-ReliefF reproduce la recuperación de ReliefF en todo el rango y Proto-ReliefF solo se separa de forma marginal en el nivel de ruido más alto. El objetivo se cumple; un barrido más amplio sobre otros tipos de ruido y más conjuntos queda como posible ampliación.

- *Analizar la escalabilidad.* La escalabilidad se aborda en dos planos. La Sección 4.3 mide el exponente de crecimiento de los cuatro algoritmos y constata que las variantes propuestas pasan de un comportamiento superlineal a uno cuasi-lineal o sublineal; la Sección 4.4 lleva las propuestas hasta 10^7 instancias sintéticas y a tres conjuntos reales masivos, verificando su viabilidad práctica.

5.3 Principales aportaciones

Más allá del cumplimiento de los objetivos, el trabajo deja varias aportaciones concretas. La primera es de carácter conceptual y resuelve la brecha identificada al cerrar el Capítulo 2. Pese a que la búsqueda aproximada de vecinos había alcanzado la madurez en otros dominios, ningún trabajo previo había integrado un índice de la generación actual en el núcleo de ReliefF. HNSW-ReliefF cubre ese hueco y demuestra que la integración no solo es posible, sino ventajosa.

La segunda aportación es la propia HNSW-ReliefF como método maduro y caracterizado. No se trata únicamente de acelerar, sino de hacerlo sin coste, ya que la equivalencia estadística con ReliefF (Sección 4.6.4) convierte la aceleración en una mejora sin contrapartida, y el

análisis de su anatomía (Sección 4.7) atribuye la ganancia a la combinación del índice con la compilación del núcleo de cómputo, identificando la contribución de cada componente.

La tercera aportación es Proto-ReliefF, concebida para el caso en el que incluso una búsqueda aproximada sobre todas las instancias resulta costosa. Su capacidad para operar con normalidad a escalas en las que ReliefF fracasa por tiempo y por memoria amplía el rango de aplicabilidad de toda la familia Relief, y lo hace además con la menor huella de memoria de las tres implementaciones.

La cuarta aportación es una guía de uso fundamentada empíricamente. El trabajo no concluye que una variante sea mejor que la otra, sino que delimita cuándo conviene cada una a partir del cruce de rendimiento observado entre ambas (Sección 4.4), HNSW-ReliefF en el rango medio, donde su índice rinde mejor, y Proto-ReliefF en el extremo masivo, donde su compresión resulta más eficiente.

La quinta aportación es la cuantificación del beneficio ambiental. El trabajo no se limita a afirmar que reducir el cómputo ahorra energía, sino que lo mide en términos de emisiones equivalentes de CO_2 y resume el compromiso entre calidad y huella en una sola magnitud, el ratio de eficiencia verde (Sección 4.5), alineando el resultado con el marco de la IA Verde [4]. Por último, como aportación transversal a la reproducibilidad, todo el código se ha liberado como software libre, lo que permite verificar los resultados y construir sobre ellos.

5.4 Discusión de los resultados

Entre todos los resultados, hay uno que sostiene al resto y conviene destacar, la equivalencia en calidad. La narrativa de eficiencia solo tiene sentido si la selección producida sigue siendo útil, y por eso el resultado habilitante del trabajo no es la aceleración, sino la comprobación de que esa aceleración no degrada la selección. El examen de las posiciones de las variables sobre un conjunto con *ground truth* conocido (Figura 4.10) es particularmente ilustrativo, pues los tres algoritmos ordenan las variables de forma prácticamente idéntica, situando las causales en cabeza y descartando el ruido. La aproximación de HNSW-ReliefF y la compresión de Proto-ReliefF alteran el coste del cálculo, pero no el juicio sobre qué variables importan.

El segundo hallazgo relevante es que las dos variantes no compiten, sino que se complementan. El cruce de rendimiento observado al llevar ambas al límite no es un accidente, sino la consecuencia directa de sus exponentes de crecimiento. El menor exponente de Proto-ReliefF, derivado de operar sobre un número acotado de prototipos, termina imponiéndose sobre el de HNSW-ReliefF, que debe construir y consultar un índice sobre el conjunto completo. Que el punto de cruce aparezca en un tamaño concreto y que la ventaja de Proto-ReliefF se amplíe con la escala confirma de forma empírica la guía de uso anticipada en el diseño, y refuerza la

idea de que ambas propuestas cubren tramos distintos de un mismo continuo de tamaños.

El tercer hallazgo, de carácter más metodológico, emerge de la descomposición de la aceleración por componentes (Sección 4.7). Analizar HNSW-ReliefF por partes revela que la ganancia no procede de un único recurso, sino de la suma de contribuciones independientes, entre las que destacan la compilación del núcleo de cómputo, el índice de proximidad y el submuestreo de instancias de consulta, que actúan sobre fases distintas del algoritmo y, por ello, se multiplican en lugar de solaparse. Igual de relevante es que ninguna de esas contribuciones degrada la selección, pues la aceleración acumulada se obtiene de forma íntegra y sin contrapartida en calidad. Este resultado refuerza, desde el nivel de los componentes, la misma conclusión que los tests estadísticos establecen de forma global, y aporta confianza en que la eficiencia del método no descansa sobre un equilibrio frágil, sino sobre decisiones de diseño ortogonales entre sí.

El cuarto hallazgo enlaza el plano técnico con el ambiental. La reducción de emisiones no es un objetivo independiente, sino una consecuencia mecánica del menor consumo de CPU, ya que un menor tiempo de cómputo se traduce, casi linealmente, en menos energía y menos emisiones. Lo destacable es que ese ahorro se obtiene sin sacrificar calidad, una combinación que en el marco de la IA Verde resulta especialmente valiosa cuando la selección de características forma parte de *pipelines* que se ejecutan de forma repetida. En ese contexto, un ahorro porcentual constante por ejecución se acumula a lo largo de muchas, y la elección de una variante eficiente deja de ser una mejora marginal para convertirse en una decisión con impacto agregado.

5.5 Limitaciones

El alcance de las conclusiones debe matizarse con varias limitaciones, algunas inherentes al enfoque y otras derivadas del protocolo experimental. La principal limitación de Proto-ReliefF es su dependencia de la calidad del *clustering*. Cuando una clase tiene una estructura multimodal o una frontera de decisión compleja, la compresión puede situar prototipos en regiones poco representativas y degradar la selección. Este efecto se observa con claridad en un único conjunto de la batería de calidad (Sección 4.6.1) y es coherente con la limitación descrita en el diseño (Sección 3.4.2); explica además casi por completo la diferencia de media de Proto-ReliefF respecto a las otras dos variantes.

HNSW-ReliefF, por su parte, hereda la sensibilidad de los grafos de proximidad a las distribuciones muy heterogéneas en densidad, donde el *recall* de la búsqueda aproximada puede decaer si la conectividad del índice no se ajusta. Aunque en la práctica este efecto no se tradujo en una pérdida de calidad apreciable, constituye un riesgo conocido en distribuciones adversas.

A las limitaciones inherentes al enfoque se suma una de naturaleza práctica. Ambas variantes introducen hiperparámetros de los que el ReliefF original carecía, como los parámetros del índice en HNSW-ReliefF o el grado de compresión en Proto-ReliefF. El trabajo mitiga ese coste mediante valores por defecto adaptativos al tamaño y la estructura del conjunto, y mediante una configuración Pareto-óptima fijada una sola vez (Sección 4.2.4), de modo que en la práctica ninguna de las dos variantes exige ajuste manual. Aun así, la superficie de configuración es mayor que la del método original, y un uso fuera de los regímenes estudiados podría requerir reajustes. A ello se añade que HNSW-ReliefF depende de que la métrica de distancia sea compatible con el índice de proximidad, una condición que se cumple en los escenarios habituales de selección de características pero que acota su aplicabilidad en espacios con métricas no estándar.

En el plano experimental, el estudio de robustez al ruido se centró en un único conjunto con *ground truth* y en el ruido de etiquetas, de modo que sus conclusiones, aunque claras dentro de ese marco, no agotan otros tipos de perturbación. Las medidas en el régimen masivo se realizaron con una sola semilla y un único *holdout*, una decisión justificada por el carácter casi determinista del tiempo de ajuste y por el coste de la validación cruzada a esa escala, pero que limita la cuantificación de la variabilidad de la calidad en ese tramo. Finalmente, la validación estadística se efectuó sobre doce conjuntos de tamaño pequeño y medio. El número de bloques pareados es suficiente para los tests empleados, pero la equivalencia establecida debe entenderse como ausencia de diferencias detectables con esa potencia, no como identidad estricta de comportamiento.

5.6 Líneas de trabajo futuro

Las limitaciones anteriores señalan de forma natural varias direcciones de continuación. La más inmediata para HNSW-ReliefF es un mecanismo de *recall* adaptativo que detecte distribuciones difíciles y ajuste automáticamente los parámetros de consulta del índice para garantizar una calidad mínima, eliminando el riesgo asociado a las densidades heterogéneas. Para Proto-ReliefF, la sustitución del *clustering* por mezclas de gaussianas o por métodos jerárquicos permitiría representar fronteras multimodales con varios prototipos por modo, atacando directamente su principal punto débil.

Una segunda dirección refuerza la solidez empírica del trabajo. Un estudio de robustez al ruido más amplio, que barra distintos tipos de perturbación y niveles de intensidad sobre más conjuntos, ampliaría el análisis de la Sección 4.6.3 y permitiría caracterizar con precisión la tolerancia de cada variante. En la misma línea, ampliar la validación estadística a más conjuntos y a datos reales de mayor tamaño aumentaría la potencia de los contrastes.

Una tercera dirección amplía el ámbito de aplicación. Ninguna de las dos propuestas con-

templa, en su forma actual, el aprendizaje incremental, ya que la llegada de nuevas instancias obliga a reconstruir el índice o a regenerar los prototipos. Adaptar ambas variantes a escenarios de *streaming*, mediante ventanas deslizantes o esquemas de actualización con descuento, abriría la puerta a la selección de características en flujos continuos de datos. A escalas todavía mayores, del orden de miles de millones de instancias, la incorporación de técnicas de cuantización vectorial o de aceleración por GPU podría extender el alcance de la búsqueda aproximada más allá de lo explorado aquí.

Por último, dado que el criterio de relevancia se preserva sin cambios, ambas estrategias son trasladables a otras tareas y variantes de la familia Relief. Su extensión al caso de regresión, sobre la base de RReliefF, o su combinación en un esquema híbrido que comprima primero el conjunto y construya después el índice sobre los prototipos, son continuaciones directas que aprovecharían simultáneamente las dos ideas desarrolladas en este trabajo.

5.7 Relación con las competencias de la titulación

El trabajo realizado integra y pone en práctica conocimientos adquiridos a lo largo de toda la titulación, hasta el punto de que puede leerse como una síntesis aplicada de buena parte de su plan de estudios.

El objeto mismo del trabajo, la selección de características y su evaluación mediante clasificadores, se asienta sobre la formación en aprendizaje automático, presente de forma escalonada en *Aprendizaje Automático I*, *II* y *III*, y muy especialmente en *Aprendizaje Automático a Gran Escala*, cuyo interés por el comportamiento de los algoritmos cuando el volumen de datos crece coincide con la motivación central de este TFG. La perspectiva sobre la alta dimensionalidad y la maldición que la acompaña, omnipresente en el planteamiento del problema, enlaza directamente con *Modelización Estadística de Datos de Alta Dimensión*.

El núcleo algorítmico se apoya en *Diseño y Análisis de Algoritmos*, cuya manera de razonar sobre el coste asintótico es la que permite enunciar y justificar el paso de una complejidad cuadrática a una cuasi-lineal. La selección de configuraciones Pareto-óptimas y el esquema de ponderación por distancia inversa se relacionan con *Optimización Matemática*, mientras que la formalización de las distancias y los espacios vectoriales sobre los que opera ReliefF tiene su base en *Álgebra Lineal* y *Cálculo Multivariable*.

La componente de eficiencia, que es lo que distingue a las variantes propuestas, bebe directamente de las materias de computación paralela. La paralelización por clases, la compilación del núcleo de cómputo y el procesamiento por lotes sensible a la jerarquía de caché se sustentan en lo aprendido en *Procesamiento Paralelo* y *Procesamiento Paralelo Avanzado*, mientras que el cuidado por la localidad de los datos y el aprovechamiento de los recursos del procesador remite a *Fundamentos de Computadores*.

La validación de los resultados descansa sobre *Inferencia Estadística*, de donde proceden los contrastes no paramétricos y el razonamiento sobre la significación y el tamaño de efecto que sostienen la conclusión de equivalencia entre las variantes propuestas y el método original.

Por último, el trabajo activa competencias menos evidentes pero igualmente formativas. El propio índice HNSW que vertebra una de las propuestas es una herramienta nacida en el ámbito de la *Recuperación de Información*, de modo que su uso aquí ilustra la transferencia de técnicas entre subdisciplinas de la ciencia de datos. La planificación temporal, la estimación de costes y la organización general del proyecto (Sección 1.2) ponen en práctica lo trabajado en *Gestión de Proyectos de Ingeniería de Datos*, y la redacción de la propia memoria y su posterior defensa ejercitan la comunicación técnica que recorre toda la titulación. En conjunto, este TFG no demuestra las competencias del grado de forma aislada, sino que las articula en un único problema real, lo que da la medida de la formación integradora que persigue.

5.8 Valoración final

El trabajo cierra el círculo que abrió la introducción. ReliefF era un método valioso pero atrapado por su coste, capaz de capturar interacciones que otros filtros no ven pero incapaz de hacerlo a la escala que los problemas reales exigen. Las dos variantes desarrolladas eliminan esa barrera por caminos distintos y complementarios, y lo hacen sin renunciar a aquello que hacía a ReliefF valioso en primer lugar, como confirma la equivalencia estadística en calidad. El método deja de ser una herramienta limitada a conjuntos pequeños para convertirse en una opción viable sobre millones de instancias.

Quizá lo más satisfactorio del resultado es que la eficiencia y la calidad no se presentan aquí como términos enfrentados. Lo habitual es tener que elegir entre métodos rápidos y métodos buenos; este trabajo muestra un caso en el que es posible acelerar un algoritmo en más de dos órdenes de magnitud, reducir su huella ambiental en más de un tercio y, aun así, no degradar lo que produce. En el contexto de una disciplina que empieza a medir su impacto energético, demostrar que un método clásico puede modernizarse para ser a la vez más rápido y más verde, sin pagar el precio en calidad, es una contribución modesta pero coherente con la dirección hacia la que avanza la inteligencia artificial sostenible.

Apéndices

Apéndice A

Reproducibilidad de los experimentos

Este anexo reúne la información necesaria para reproducir los experimentos del Capítulo 4, en coherencia con el compromiso de ciencia abierta del trabajo. Todo el código, los conjuntos sintéticos y los *scripts* de experimentación están disponibles en el repositorio público del proyecto[1], liberado como software libre.

A.1 Repositorio y organización del código

El repositorio se estructura en cuatro bloques. El primero es el paquete con la implementación de HNSW-ReliefF y Proto-ReliefF, junto con las utilidades de preprocesado, generación de prototipos y medición. El segundo agrupa los *scripts* que reproducen cada análisis del capítulo, esto es, la eficiencia, la escalabilidad al límite, la huella de carbono, la preservación de la calidad y la anatomía de la aceleración. El tercero contiene la configuración de hiperparámetros, empleada de forma uniforme en toda la experimentación. El cuarto recoge las salidas numéricas y las figuras generadas, que son las incorporadas a esta memoria.

A.2 Entorno de ejecución

Todas las mediciones se realizaron en el entorno software recogido en la Tabla 4.4 y sobre un procesador AMD Ryzen 7 7800X3D (8 núcleos y 16 hilos), 48 GB de memoria DDR5 y Windows 11. Las dependencias son de código abierto y se instalan en un entorno virtual de Python 3.11. Para garantizar resultados idénticos conviene fijar las versiones de las librerías, en especial las de *scikit-learn*, *hnswlib*, *Numba* y *codecarbon*, de las que dependen tanto la búsqueda de vecinos como la medición de tiempo y de emisiones.

[1] https://github.com/nicolasallerponte/ReliefF-Optimization

A.3 Configuración de los algoritmos

Los experimentos emplean las configuraciones Pareto-óptimas determinadas en la búsqueda de hiperparámetros (Sección 4.2, Tabla 4.6), almacenadas en un único fichero de configuración:

```
HNSW-ReliefF:
  n_neighbors:      10
  M:                8
  ef_construction:  100
  ef_search:        50
Proto-ReliefF:
  k_protos:         15
  sigma:            0.15
  use_lvq:          false
```

Los valores no fijados de forma explícita, como la fracción de muestreo en HNSW-ReliefF o el número de prototipos en Proto-ReliefF, se calculan de manera adaptativa según el tamaño y la estructura de cada conjunto, tal como se describe en el Capítulo 3.

A.4 Protocolo experimental

El protocolo completo se detalla en la Sección 4.1.2. A efectos de reproducibilidad, los elementos que fijan la aleatoriedad y la comparación son los siguientes:

- Semillas: 42, 123, 456, 789 y 1234.
- Validación: cinco pliegues estratificados repetidos sobre las cinco semillas, es decir, 25 evaluaciones por celda, en los conjuntos de tamaño pequeño y medio; un único *holdout* en los conjuntos reales masivos, donde la validación cruzada es inviable.
- Clasificador posterior: RandomForest de 100 árboles y profundidad máxima 10, entrenado sobre las 10 variables de mayor relevancia.
- Tiempo: medido con una única semilla, por su carácter prácticamente determinista.
- Emisiones: medidas con codecarbon a partir del consumo de CPU (Sección 4.5).

A.5 Ejecución

Una vez instaladas las dependencias, cada análisis del capítulo puede reproducirse ejecutando el *script* correspondiente, y el repositorio incluye además otro que lanza la batería completa. Las figuras y tablas resultantes se depositan en la carpeta de resultados, desde donde se han incorporado a esta memoria. Al fijar las semillas y las versiones de las dependencias, una nueva ejecución reproduce los valores de calidad presentados; los tiempos absolutos pueden variar según el *hardware*, pero no así las tendencias de escala ni las conclusiones del trabajo.

Lista de acrónimos

ANN Approximate Nearest Neighbors. 2

AUC Area Under the Curve. 5

CPU Central Processing Unit. 42

GMM Gaussian Mixture Model. 39

GPU Graphics Processing Unit. 69

HNSW Hierarchical Navigable Small World. 2

IA Inteligencia Artificial. 1

IQR Interquartile Range. 30

IVF Inverted File. 15

JIT Just-In-Time. 24

LSH Locality-Sensitive Hashing. 15

LVQ Learning Vector Quantization. 19

RAM Random Access Memory. 14

RAPL Running Average Power Limit. 51

SIMD Single Instruction, Multiple Data. 27

SURF Spatially Uniform ReliefF. 12

TFG Trabajo de Fin de Grado. 2

UDC Universidade da Coruña. 2

Bibliografía

[1] Zilliz, "Hierarchical navigable small worlds (HNSW)," https://zilliz.com/learn/hierarchical-navigable-small-worlds-HNSW, 2025, accedido: junio de 2026.

[2] R. Bellman, "Dynamic programming," *science*, vol. 153, no. 3731, pp. 34–37, 1966.

[3] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature extraction: foundations and applications*. Springer Science & Business Media, 2006, vol. 207.

[4] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[5] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 3645–3650.

[6] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of bloom, a 176b parameter language model," *Journal of machine learning research*, vol. 24, no. 253, pp. 1–15, 2023.

[7] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[8] Y. Tan, G. Long, L. Liu, T. Zhou, Q. Lu, J. Jiang, and C. Zhang, "Fedproto: Federated prototype learning across heterogeneous clients," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 8432–8440.

[9] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[10] E. M. Karabulut, S. A. Özel, and T. Ibrikci, "A comparative study on the effect of feature selection on classification accuracy," *Procedia Technology*, vol. 1, pp. 323–327, 2012.

[11] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[12] A. Jakulin and I. Bratko, "Analyzing attribute dependencies," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2003, pp. 229–240.

[13] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 121–129.

[14] R. J. Urbanowicz, M. Meeker, W. La Cava, R. S. Olson, and J. H. Moore, "Relief-based feature selection: Introduction and review," *Journal of Biomedical Informatics*, vol. 85, pp. 189–203, 2018.

[15] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Machine Learning Proceedings 1992*. Elsevier, 1992, pp. 249–256.

[16] I. Kononenko, "Estimating attributes: Analysis and extensions of RELIEF," in *European Conference on Machine Learning*. Springer, 1994, pp. 171–182.

[17] M. Robnik-Šikonja and I. Kononenko, "Theoretical and empirical analysis of ReliefF and RReliefF," *Machine learning*, vol. 53, no. 1, pp. 23–69, 2003.

[18] R. J. Urbanowicz, R. S. Olson, P. Schmitt, M. Meeker, and J. H. Moore, "Benchmarking relief-based feature selection methods," *Journal of Biomedical Informatics*, vol. 85, pp. 168–188, 2018.

[19] C. S. Greene, N. M. Penrod, J. Kiralis, and J. H. Moore, "Spatially Uniform ReliefF (SURF) for computationally-efficient filtering of gene-gene interactions," *BioData Mining*, vol. 2, no. 1, p. 5, 2009.

[20] C. S. Greene, D. S. Himmelstein, J. Kiralis, and J. H. Moore, "The informative extremes: Using both nearest and farthest individuals can improve Relief algorithms in the domain of human genetics," in *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Springer, 2010, pp. 182–193.

[21] D. Granizo-MacKenzie and J. H. Moore, "Multiple threshold spatially uniform ReliefF for the genetic analysis of complex human diseases," in *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Springer, 2013, pp. 1–10.

[22] J. H. Moore and B. C. White, "Tuning ReliefF for genome-wide genetic analysis," in *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Springer, 2007, pp. 166–175.

[23] X. Huang, L. Zhang, B. Wang, F. Li, and Z. Zhang, "An optimization of ReliefF for classification in large datasets," *Pattern Recognition Letters*, 2010.

[24] R.-J. Palma-Mendoza, D. Rodriguez, and L. de-Marcos, "Distributed ReliefF-based feature selection in Spark," *Knowledge and Information Systems*, vol. 57, no. 1, pp. 1–20, 2018.

[25] Z. Abbasi and M. Rahmani, "An instance selection algorithm based on ReliefF," *International Journal on Artificial Intelligence Tools*, vol. 28, no. 1, p. 1950001, 2019.

[26] Z. Abbasi, M. Rahmani, and H. Ghaffarian, "IFSB-ReliefF: A new instance and feature selection algorithm based on ReliefF," *Journal of Signal and Data Processing (JSDP)*, vol. 17, no. 4, pp. 49–64, 2021.

[27] S. Xu, X. Li, and W. F. Lu, "Randomized kd tree relieff algorithm for feature selection in handling high dimensional process parameter data," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–8.

[28] M. Wang, X. Xu, Q. Yue, and Y. Wang, "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 1964–1978, 2021.

[29] M. Aumüller, E. Bernhardsson, and A. Faithfull, "ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," in *International Conference on Similarity Search and Applications*. Springer, 2017, pp. 34–49.

[30] Y. Liu *et al.*, "Efficient approximate nearest neighbor search via data-adaptive parameter adjustment in hierarchical navigable small graphs," *IEEE access : practical innovations, open solutions*, 2025.

[31] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE transactions on big data*, vol. 7, no. 3, pp. 535–547, 2019.

[32] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar, "Accelerating large-scale inference with anisotropic vector quantization," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3887–3896.

[33] M. Aumüller, E. Bernhardsson, and A. Faithfull, "ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," *Information Systems*, vol. 87, 2020.

[34] D. Karapiperis, L. Akritidis, P. Bozanis, and V. S. Verykios, "A comparative analysis of graph-based and partition-based approximate nearest neighbor search for large-scale entity resolution," *Intelligent Decision Technologies*, vol. 19, no. 6, pp. 3826–3840, 2025.

[35] O. P. Elliott and J. Clark, "The impacts of data, ordering, and intrinsic dimensionality on recall in hierarchical navigable small worlds," in *Proceedings of the 2024 ACM SIGIR International Conference on Theory of Information Retrieval*, 2024, pp. 25–33.

[36] J. B. McQueen, "Some methods of classification and analysis of multivariate observations," in *Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.*, 1967, pp. 281–297.

[37] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 1177–1178.

[38] K. Leonard and P. Rousseeuw, "Partitioning around medoids (program pam)," *Finding Groups in Data; John Wiley and Sons, Ltd.: Hoboken, NJ, USA*, pp. 68–125, 1990.

[39] T. Kohonen, "Learning vector quantization," in *Self-organizing maps*. Springer, 1995, pp. 245–261.

[40] A. S. Banait, P. G. Fegade, R. Gawande, R. B. Mandlik, J. A. Kandekar, and S. S. Banait, "Adaptive data sampling and feature optimization with LSH-IS and Pearson correlation in big data contexts," *ANVI Journal of Scholarly Research and Innovation*, vol. 27, 2024.

[41] L. Sun, Q. Zhang, W. Ding, T. Wang, and J. Xu, "Fcpfs: fuzzy granular ball clustering-based partial multilabel feature selection with fuzzy mutual information," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 9, no. 1, pp. 590–606, 2024.

[42] L. Peng and W. Shu, "Multi-label feature selection based on granular ball and fuzzy neighborhood mutual information," in *Proceedings of the 2024 4th International Conference on Internet of Things and Machine Learning*, 2024, pp. 260–266.

[43] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[46] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.