



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Mejora de la eficiencia computacional de un sistema de
predicción de tipo de buque utilizando parámetros de trayectorias y
técnicas de IA*

Grado en Ingeniería Mecánica

ALUMNO: Francisco Javier Pavón Jarava

DIRECTORES: Belén Barragáns Martínez
Pablo Sendín Raña

CURSO ACADÉMICO: 2024-2025

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Mejora de la eficiencia computacional de un sistema de
predicción de tipo de buque utilizando parámetros de trayectorias y
técnicas de IA*

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

UniversidadeVigo

RESUMEN

El presente Trabajo de Fin de Grado se enmarca en la continuación de una línea de investigación centrada en la predicción del tipo de buque a partir de parámetros derivados de sus trayectorias, utilizando técnicas de aprendizaje automático. Los trabajos previos ofrecían resultados prometedores en cuanto a precisión, pero presentaban limitaciones computacionales importantes, con tiempos de ejecución que podían superar las 70 horas, lo que dificultaba su aplicación en entornos operativos o en análisis de gran volumen de datos.

Este TFG tiene como objetivo principal mejorar la eficiencia computacional del sistema de predicción, manteniendo su capacidad clasificadora. Para ello, se han rediseñado varias fases del procesamiento, eliminando el acceso iterativo a los datos y unificando la carga en un único DF. Además, se ha migrado el procesamiento a GPU mediante el uso de *cuDF*, lo que permite aprovechar el paralelismo masivo para reducir significativamente los tiempos de ejecución.

También se ha analizado la relevancia de los parámetros dinámicos, con el fin de reducir la complejidad del modelo eliminando aquellas variables con escasa contribución predictiva. Estas optimizaciones sientan las bases para un sistema más ligero, escalable y adaptable a aplicaciones en tiempo real o en entornos con recursos computacionales limitados.

PALABRAS CLAVE

Inteligencia Artificial, AIS, Tipo de buque, Paralelización, GPU, *cuDF*.

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a todo el personal de la Escuela Naval Militar por su esfuerzo y tiempo invertido en mí y en todos mis compañeros, con una firme vocación y ejemplaridad a lo largo de estos cinco años, que nos ha ayudado a desarrollarnos como militares y personas.

Agradecer también la incansable labor de mis tutores, Belén y Pablo, que con tanta energía y alegría me han guiado en los últimos meses apoyando y solventando todos los baches del camino. Sin vuestro apoyo, el camino habría sido mucho más difícil.

También agradecer a mi familia y, sobre todo, a mis padres y hermanas quienes han hecho de mí la persona que soy. Todo su apoyo y largos viajes por venir al otro rincón de España me han hecho feliz y me han dado fuerzas para ser constante durante estos cinco años y quienes me han enseñado que todo esfuerzo tiene su recompensa.

A mis amigos, tanto aquellos compañeros de la promoción 425-155 que me han acompañado en este viaje y han hecho de la Escuela una experiencia de vida y que sea tan especial como ha sido, como a todos ellos que, desde El Puerto de Santa María me han demostrado su amistad y apoyo constante. No podría olvidarme de mencionar el apoyo prestado por Pilar, quien diariamente me ha ayudado a ver el lado bueno de las cosas y a dar un poco más día tras día sacando lo mejor de mí. A mi camareta que ha hecho de este año, un año inolvidable.

A todos ellos, muchas gracias por vuestro apoyo incondicional.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	7
1 Introducción y objetivos	9
1.1 Contextualización.....	9
1.2 Motivación	9
1.3 Objetivos	10
1.4 Organización de la memoria	10
2 Estado del arte	11
2.1 Inteligencia Artificial	11
2.2 <i>Machine Learning</i>	12
2.2.1 Aprendizaje supervisado.....	13
2.2.2 Aprendizaje no supervisado.....	14
2.2.3 Aprendizaje semisupervisado	16
2.2.4 Aprendizaje por refuerzo	17
2.2.5 <i>Deep Learning</i>	18
2.2.6 Campos de aplicación	20
2.3 Algoritmos de aprendizaje automático supervisado	21
2.3.1 Árboles de decisión.....	22
2.3.2 <i>Random Forest</i>	23
2.3.3 <i>XGBoost</i>	24
2.4 Conocimiento del Entorno Marítimo	25
2.4.1 Estrategia Nacional de Seguridad Marítima	27
2.5 Trabajos previos	30
3 Desarrollo del TFG.....	33
3.1 Entorno de trabajo y herramientas empleadas	33
3.1.1 Hardware.....	33
3.1.2 <i>Jupyter Lab</i>	34
3.1.3 <i>Python</i>	34
3.1.5 <i>cuDF</i>	35
3.2 Datos estáticos.....	35
3.2.1 Definición de los campos de los datos estáticos y parámetros extraídos.....	36
3.2.3 Experimentos estáticos	39

3.3 Datos dinámicos	42
3.3.1 Definición de los campos de los datos dinámicos y parámetros extraídos	43
3.3.2 Detección de cuellos de botella	43
3.3.3 Experimentos dinámicos.....	44
4 Resultados obtenidos.....	71
4.1 Estructuración de los experimentos	71
4.2 Análisis de los resultados	73
4.2.1 Mejora de la calidad de predicción en los datos estáticos con <i>XGBoost</i>	73
4.2.2 Resultados del Experimento datos globales 0	75
4.2.3 Aplicación de la directiva <i>%load_ext cudf.pandas</i>	75
4.2.4 Primera iteración.....	75
4.2.5 Segunda iteración.....	79
4.2.6 Optimización del acceso a los datos	82
5 Conclusiones y líneas futuras	89
5.1 Conclusiones	89
5.2 Líneas futuras	91
6 Bibliografía.....	93
Anexo I: Implicaciones Sociales, Económicas y Ambientales	97
Anexo II: Reflexiones Éticas y Sociales	99
Anexo III: Código desarrollado.....	101
Anexo IV: Carga y procesado unificado de los datos	121
Anexo V: Resultados en la evaluación de los modelos.....	127

ÍNDICE DE FIGURAS

Figura 2-1 Esquema de aprendizaje supervisado [11]	14
Figura 2-2 Esquema de aprendizaje no supervisado [14]	15
Figura 2-3 Ejemplo de funcionamiento de aprendizaje no supervisado [15]	15
Figura 2-4 Estructura de funcionamiento de aprendizaje semisupervisado [12]	17
Figura 2-5 Representación de los componentes y relaciones entre sí en el aprendizaje por refuerzo [18]	18
Figura 2-6 Relación entre la inteligencia artificial, ML y <i>Deep Learning</i> [21].....	19
Figura 2-7 Comparación del funcionamiento de sistemas de ML y DL [21]	19
Figura 2-8 Ejemplo de árbol de decisión [26].....	22
Figura 2-9 Ejemplo de predicción mediante <i>Random Forest</i> [30].....	24
Figura 2-10 Ejemplo de una RMP sin clasificar [35]	26
Figura 2-11 Espacios marítimos [37].....	27
Figura 2-12 Espacios marítimos de interés [12]	28
Figura 2-13 Sala del COVAM desde donde se controlan las aguas españolas o de especial interés [36]	29
Figura 3-1 Pila de <i>hardware</i> y <i>software</i> optimizada para el proceso de ciencia de datos con RAPIDS [52]	35
Figura 3-2 Almacenamiento de los datos de estáticos en <i>df_raw_static</i>	36
Figura 3-3 Obtención de precisión y métricas	38
Figura 3-4 Resultados obtenidos con <i>Random Forest</i> de [1].....	39
Figura 3-5 Resultados del <i>Experimento estáticos 0</i>	39
Figura 3-6 Configuración del algoritmo <i>XGBoost</i>	40
Figura 3-7 Resultados del <i>Experimento estáticos 1</i>	41
Figura 3-8 Resultados del <i>Experimento estáticos 2</i>	41
Figura 3-9 Resultados del <i>Experimento estáticos 3</i>	42
Figura 3-10 Resultados del <i>Experimento estáticos 4</i>	42
Figura 3-11 Tiempos de ejecución del procesamiento de datos dinámicos en el <i>Experimento datos globales 0</i>	44
Figura 3-12 Funcionamiento de la directiva <i>%load_ext cudf.pandas</i> [58]	45
Figura 3-13 Creación de <i>df_raw_dynamic</i>	46
Figura 3-14 Creación de <i>df_longitudes</i>	47
Figura 3-15 Creación del DF <i>ejemplo_merged</i>	48
Figura 3-16 Representación gráfica de los puntos GPS de un buque	48
Figura 3-17 Trayectoria del <i>Tanker 1</i>	49
Figura 3-18 Trayectoria del <i>Cargo 1</i>	49

Figura 3-19 Trayectoria del <i>Pesquero 1</i>	49
Figura 3-20 Implementacion de la fórmula de <i>Haversine</i> en <i>Python</i>	51
Figura 3-21 Implementación de la función <i>calcular_trayectorias</i>	52
Figura 3-22 Implementación de la función <i>detectar_paradas_cudf</i>	52
Figura 3-23 Implementación de la función <i>detectar_segmentos_parada_cudf</i>	53
Figura 3-24 Implementación de la función <i>dividir_trayectorias_por_paradas</i>	53
Figura 3-25 Creación del DF <i>df_merged_dynamic</i>	54
Figura 3-26 Diagrama de flujo del bucle de extracción de parámetros	55
Figura 3-27 Implementación de la función <i>clean</i>	56
Figura 3-28 Aplicación de las funciones de detección y discretización de trayectorias	56
Figura 3-29 Implementación de la función <i>calcular_parametros_trayectoria</i>	58
Figura 3-30 Implementación de la función <i>calcular_parametros_paradas</i>	59
Figura 3-31 Declaración de parámetros dinámicos a extraer en la primera iteración.....	62
Figura 3-32 Gráfica sobre la importancia de los parámetros en la predicción	64
Figura 3-33 Primera prueba preliminar para eliminar columnas en la segunda iteración en <i>Random Forest</i>	64
Figura 3-34 Segunda prueba preliminar para eliminar columnas en la segunda iteración en <i>Random Forest</i>	65
Figura 3-35 Resultados de la segunda prueba preliminar de <i>Random Forest</i>	65
Figura 3-36 Declaración de parámetros dinámicos a extraer en la segunda iteración	65
Figura 3-37 Prueba preliminar para eliminar columnas en la segunda iteración en <i>XGBoost</i>	66
Figura 3-38 Resultados de la prueba preliminar de <i>XGBoost</i>	66
Figura 3-39 Implementación de la función <i>clean</i> para optimizar el acceso a los datos.....	68
Figura 3-40 Implementación del código para la detección de paradas	69
Figura 3-41 Implementación de la lectura y llamadas a funciones sin iteración por MMSI	70
Figura 4-1 Resultados de la predicción en estáticos con <i>XGBoost (cuDF)</i>	73
Figura 4-2 Resultados de la predicción en estáticos con <i>XGBoost (pandas)</i>	74
Figura 4-3 Tiempos de ejecución con la directiva <i>%load_ext cudf.pandas</i>	75
Figura 4-4 Tiempos de ejecución de la primera iteración con <i>cuDF</i>	76
Figura 4-5 Resultados del <i>Experimento datos globales 1</i> con <i>Random Forest</i>	76
Figura 4-6 Resultados del <i>Experimento datos globales 1</i> con <i>XGBoost</i>	77
Figura 4-7 Resultados del <i>Experimento datos globales 2</i> con <i>Random Forest</i>	78
Figura 4-8 Resultados del <i>Experimento datos globales 2</i> con <i>XGBoost</i>	79
Figura 4-9 Tiempo de ejecución de la segunda iteración, 29 parámetros eliminados para <i>Random Forest</i>	79
Figura 4-10 Tiempo de ejecución de la segunda iteración, 42 parámetros eliminados para <i>Random Forest</i>	79

Figura 4-11 Tiempos de ejecución tras la reducción de parámetros para <i>XGBoost</i>	80
Figura 4-12 Resultados del <i>Experimento datos globales 3</i> con <i>Random Forest</i>	80
Figura 4-13 Resultados del <i>Experimento datos globales 3</i> con <i>XGBoost</i>	81
Figura 4-14 Resultados del <i>Experimento datos globales 4</i> con <i>Random Forest</i>	82
Figura 4-15 Resultados del <i>Experimento datos globales 4</i> con <i>XGBoost</i>	82
Figura 4-16 Tiempo de ejecución del código de procesamiento de datos en <i>df_dynamic</i>	83
Figura 4-17 Resultados del <i>Experimento datos globales 5</i> con <i>Random Forest</i>	84
Figura 4-18 Resultados del <i>Experimento datos globales 5</i> con <i>XGBoost</i>	85
Figura 4-19 Resultados del <i>Experimento datos globales 6</i> con <i>Random Forest</i>	85
Figura 4-20 Resultados del <i>Experimento datos globales 6</i> con <i>XGBoost</i>	86
Figura 4-21 Gráfica de densidad del parámetro <i>ldivw</i>	87
Figura 4-22 Gráficas de densidad para distintos parámetros	88

ÍNDICE DE TABLAS

Tabla 2-1 Los cuatro enfoques de Russell y Norvig para la inteligencia artificial [7]	12
Tabla 2-2 Comparación entre <i>Random Forest</i> y árbol de decisión [30]	24
Tabla 3-1 Campos del fichero de datos estáticos (csv)	37
Tabla 3-2 Descripción de los parámetros estáticos	37
Tabla 3-3 Mapeo de <i>shiptype</i> de <i>string</i> a código numérico	41
Tabla 3-4 Campos de los datos dinámicos proporcionados en el archivo csv	43
Tabla 3-5 Migración de funciones en <i>movingpandas</i> a funciones programadas explícitamente en <i>cuDF</i>	51
Tabla 4-1 Resumen de los experimentos realizados con los datos estáticos	72
Tabla 4-2 Resumen de los experimentos realizados con los datos dinámicos y la calidad de la predicción asociada a cada experimento con los parámetros globales	72
Tabla 4-3 Comparación de los tiempos de ejecución en el procesamiento de los datos dinámicos	83

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Contextualización

La seguridad marítima es una prioridad estratégica para España, dada su posición privilegiada entre Europa, África y América, situándola como un punto clave en las rutas marítimas internacionales. La dependencia del transporte marítimo es vital para la economía nacional ya que, aproximadamente, el 80% de las importaciones y el 50% de las exportaciones se realizan por vía marítima. Sin embargo, esta dependencia también implica una vulnerabilidad significativa ante amenazas como la piratería, el terrorismo, el tráfico ilícito o la pesca ilegal.

En respuesta a estos desafíos, España desarrolla estrategias enfocadas hacia la mejora del Conocimiento del Entorno Marítimo (CEM), que se materializan en organismos como el Centro de Operaciones y Vigilancia de Acción Marítima (COVAM). Este centro utiliza datos provenientes del Sistema de Identificación Automática (AIS), que permite identificar y monitorizar buques en tiempo real. Sin embargo, la fiabilidad de los datos AIS se ve afectada por errores humanos, especialmente en la clasificación manual del tipo de buque, lo cual dificulta obtener predicciones precisas y rápidas.

Investigaciones anteriores desarrolladas en el Centro Universitario de la Defensa en la Escuela Naval Militar (CUD-ENM) han abordado este problema aplicando técnicas avanzadas de Inteligencia Artificial (IA), específicamente modelos de aprendizaje automático supervisado, para mejorar la precisión en la clasificación del tipo de buque.

Sin embargo, estos modelos presentan un importante desafío computacional. Los trabajos previos realizados, como el [1], lograron crear modelos predictivos eficaces, aunque con tiempos de ejecución considerablemente largos, oscilando entre 72 y 96 horas. Este alto coste computacional impide una aplicación eficiente y práctica del sistema en entornos reales.

1.2 Motivación

La principal motivación detrás de este Trabajo Fin de Grado (TFG) radica en la necesidad de mejorar la eficiencia computacional del sistema previamente desarrollado. La elevada duración de las tareas de procesamiento representa una barrera significativa para la implementación efectiva de este sistema en entornos reales. Es esencial, por tanto, reducir estos tiempos drásticamente para permitir una predicción rápida, precisa y operativamente útil del tipo de buque.

Ante esta coyuntura, surge el planteamiento de optimizar los procesos mediante la utilización de recursos de procesamiento paralelo, aprovechando el potencial de las GPU y tecnologías especializadas. Esta estrategia busca asegurar un análisis casi en tiempo real, que pueda integrarse de manera efectiva

en los flujos de trabajo del COVAM (Centro de Operaciones y Vigilancia de Acción Marítima) y otros organismos responsables de la vigilancia marítima.

1.3 Objetivos

El objetivo principal de este trabajo es: optimizar la eficiencia computacional del sistema de predicción del tipo de buque mediante el uso de GPU, utilizando bibliotecas especializadas, que permitan reducir los tiempos de procesamiento, que originalmente podían alcanzar hasta 96 horas.

Este objetivo principal se desglosa en objetivos específicos:

1. Adaptar el código existente, evitando los cuellos de botella y compatibilidad del procesamiento, asegurando la extracción correcta de los mismos parámetros.
2. Implementar métodos específicos eficientes para la generación y discretización de trayectorias marítimas.
3. Validar los resultados mediante métricas estándar, asegurando que la optimización no comprometa la precisión de las predicciones.

1.4 Organización de la memoria

La presente memoria se estructura de forma que se facilite la comprensión integral del trabajo realizado. Tras esta introducción inicial, el documento continúa con los siguientes capítulos:

- Capítulo 2: Estado del arte. Se realiza una revisión exhaustiva de temáticas relacionadas directamente con este TFG como Inteligencia Artificial, *Machine Learning*, incluyendo sus diferentes tipos de aprendizaje (supervisado, no supervisado, semisupervisado, por refuerzo y *Deep Learning*). También se presentan los algoritmos específicos empleados en este TFG (Árboles de decisión, *Random Forest*, *XGBoost*), el contexto operacional del entorno marítimo (CEM, Estrategia Nacional de Seguridad Marítima) y una revisión de estudios anteriores relevantes para este trabajo.
- Capítulo 3: Desarrollo del TFG. Se detalla exhaustivamente la metodología utilizada de adaptación o de reescritura del código, comenzando con el entorno de trabajo y las herramientas empleadas. Se describen los procesos y experimentos realizados sobre datos estáticos y dinámicos, destacando especialmente la detección y resolución de cuellos de botella mediante paralelización y optimización computacional.
- Capítulo 4: Resultados obtenidos. Se presentan los resultados obtenidos tras implementar las mejoras propuestas. Se analizan estos resultados mediante gráficos, tablas y métricas de rendimiento, destacando la mejora en tiempos de ejecución y precisión de las predicciones. También se comparan estos resultados con el estado inicial del sistema.
- Capítulo 5: Conclusiones y líneas futuras. Se sintetizan las conclusiones obtenidas del desarrollo del TFG, revisando la consecución de los objetivos planteados y proponiendo líneas futuras que permitan profundizar en esta área de investigación y ampliar las aplicaciones del sistema desarrollado.

Finalmente, se complementa la memoria con referencias bibliográficas y anexos, incluyendo aspectos éticos, sociales, económicos, ambientales, así como el código desarrollado, permitiendo al lector obtener una visión integral del alcance y profundidad del trabajo realizado.

2 ESTADO DEL ARTE

El presente capítulo presenta en qué consiste la Inteligencia Artificial (IA), sus últimos avances y sus aplicaciones en las Fuerzas Armadas, para seguir con el estudio de las distintas técnicas de aprendizaje automático abordando los tipos de aprendizaje que incluye, tales como: el aprendizaje supervisado, aprendizaje no supervisado, aprendizaje por refuerzo, entre otros.

Por otro lado, se abordará el Conocimiento del Entorno Marítimo, la misión de la Armada en este ámbito y la motivación de un sistema de predicción de tipo de buque, del que se pretende mejorar la eficiencia.

Por último, se hará una revisión bibliográfica de aquellos TFG que preceden a éste y que forman parte de la misma línea de investigación.

2.1 Inteligencia Artificial

Cuando hablamos de IA es fácil pensar en esas películas de ciencia ficción en las que robots tomaban el control y se sobreponían a la humanidad, pero ante la pregunta de qué es realmente la Inteligencia Artificial surgen muchas definiciones. Esto no es casualidad, la Inteligencia Artificial recibe muchas definiciones debido a que es una tecnología emergente y cambiante y cuyo sentido depende del significado de la inteligencia, término que en sí mismo ya cuenta con muchas definiciones.

En primer lugar, acudiendo a la Real Academia Española (RAE) para consultar la definición de IA, se obtiene como respuesta: *“Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico”* [2]. Comparando esta definición proporcionada por la RAE en enero de 2025, se observa que ha variado con respecto a la reseñada en [3], TFG del curso 22-23 en el que se recoge que la RAE define la inteligencia artificial como *“la atribuida a las máquinas capaces de hacer operaciones propias de los seres inteligentes (con capacidad de entender o comprender)”*. Esto refuerza la idea de las diversas definiciones que recibe la Inteligencia Artificial al ser una tecnología emergente y cambiante.

Fue en 1943 cuando el médico de la Universidad de Illinois Warren McCulloch y el matemático Walter Pitts presentaron su modelo de neuronas artificiales, considerada la primera inteligencia artificial, aun cuando dicho término ni siquiera existía [4].

John McCarthy, miembro del departamento de Ciencias Informáticas de la Universidad de Stanford, durante la conferencia de Dartmouth acuñó por primera vez el término de Inteligencia Artificial y lo definió como *“la ciencia e ingenio de hacer máquinas inteligentes”* [5].

El problema con la definición de la Inteligencia Artificial radica en que sigue dependiendo del sentido que se le otorgue a la propia inteligencia, haciendo que sea tan complejo de definir como la inteligencia humana. Con el paso del tiempo, la mejora computacional y los avances tecnológicos se puede considerar que la Inteligencia Artificial es un campo de la informática enfocado en la creación de sistemas que puedan realizar tareas que requieran de inteligencia humana, véase el aprendizaje, el razonamiento y la percepción [6].

El concepto definido por los informáticos Peter Norvig y Stuart Russell mediante los cuatro enfoques redactados en su libro *Artificial Intelligence: A Modern Approach* [7] es actualmente el más aceptado. En este libro, los autores juntaron las definiciones de anteriores libros de texto y las reflejaron, como se puede ver en la Tabla 2-1, extraída del mismo libro. Por un lado, la primera fila hace referencia a aquellas inteligencias artificiales creadas con el fin de pensar por sí mismas, mientras que la segunda fila hace referencia actuaciones o comportamientos. Por otro lado, la primera columna trata de imitar a los humanos, mientras que la segunda trata de ir más allá como máquinas racionales.

Sistemas que piensan como humanos	Sistemas que piensan racionalmente
«El nuevo y excitante esfuerzo de hacer que los computadores piensen... máquinas con mentes, en el más amplio sentido literal». (Haugeland, 1985) «[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje...» (Bellman, 1978)	«El estudio de las facultades mentales mediante el uso de modelos computacionales». (Charniak y McDermott, 1985) «El estudio de los cálculos que hacen posible percibir, razonar y actuar». (Winston, 1992)
Sistemas que actúan como humanos	Sistemas que actúan racionalmente
«El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren de inteligencia». (Kurzweil, 1990) «El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor». (Rich y Knight, 1991)	«La Inteligencia Computacional es el estudio del diseño de agentes inteligentes». (Poole et al., 1998) «IA... está relacionada con conductas inteligentes en artefactos». (Nilsson, 1998)

Tabla 2-1 Los cuatro enfoques de Russell y Norvig para la inteligencia artificial [7]

Por último, dentro la Inteligencia Artificial existe una clasificación que la divide en líneas generales en tres categorías dependiendo de sus capacidades: la IA débil, que permite diseñar y programar ordenadores que precisen de inteligencia para realizar actividades específicas y limitadas, la IA fuerte que hace referencia a la ciencia que permite replicar la inteligencia humana con máquinas y la superinteligencia artificial. La superinteligencia artificial (ASI, *Artificial Superintelligence*) se define como una forma hipotética de inteligencia artificial general cuyas habilidades para resolver problemas son más amplias y enormemente superiores a las de cualquier contraparte humana o incluso a las de cualquier colectivo humano en un momento dado. Suponemos aquí que la velocidad evolutiva de una ASI frente a cualquier contraparte humana también es vastamente superior, lo cual parece razonable para la mayoría de las tareas de decisión en términos de rendimiento [8].

2.2 Machine Learning

El *Machine Learning* (ML, aprendizaje automático en su traducción al castellano) es la rama de la IA que se centra en la creación de sistemas a los que se les otorga capacidad de aprendizaje. El objetivo

del ML es que las máquinas aprendan ciertas tareas por sí mismas, sin necesidad de ser programadas específicamente para ellas, es decir, que las aprendan sin la necesidad de una intervención humana. Para ello, estas máquinas utilizan la estadística para predecir y reconocer patrones, por lo que disponer de grandes conjuntos de datos se convierte en una prioridad [9].

La capacidad de aprendizaje del ML depende de las diferentes técnicas y algoritmos subyacentes. Al utilizar esta tecnología, se logra una mejora significativa en la eficiencia de tareas que involucran grandes volúmenes de datos, sin que sea necesaria la intervención manual de una persona. Esto permite automatizar procesos y obtener resultados de manera más rápida y precisa. Lo que hace especial a estos algoritmos es la capacidad de modificar su comportamiento basándose en la experiencia de acciones pasadas. Existe un gran abanico de algoritmos mediante los cuales conseguir el aprendizaje automático. Podemos diferenciar en tres grandes grupos según la forma de proceder del propio algoritmo: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Más allá, existe también el *Deep Learning* explicado posteriormente en el apartado 2.2.5.

2.2.1 Aprendizaje supervisado

El aprendizaje supervisado se caracteriza por el uso de conjuntos de datos etiquetados para entrenar algoritmos capaces de clasificar datos o realizar predicciones con precisión. Durante el entrenamiento, el modelo ajusta iterativamente sus parámetros (o ponderaciones) basándose en los datos de entrada y las etiquetas asociadas, optimizándose hasta alcanzar un desempeño adecuado. Este proceso suele incluir técnicas como la validación cruzada, que ayudan a prevenir problemas comunes como el sobreajuste o el infraajuste, garantizando un modelo más robusto y generalizable.

El aprendizaje supervisado tiene aplicaciones prácticas en la resolución de problemas del mundo real a gran escala, como la detección de spam en correos electrónicos, la clasificación de imágenes o la predicción de comportamientos. Entre los métodos más utilizados en este tipo de aprendizaje se encuentran [10]:

1. Redes neuronales, que no son exclusivas del aprendizaje supervisado, son ideales para problemas complejos con datos no lineales.
2. Regresión lineal y logística, ampliamente empleadas en análisis predictivo.
3. Bosque aleatorio, que combina múltiples árboles de decisión para mejorar la precisión.
4. Máquinas de vectores de soporte (SVM, *Support Vector Machines*), eficientes en problemas de clasificación con datos de alta dimensión.

Desglosando el aprendizaje supervisado, existen dos tipos principales de aprendizaje: clasificación y regresión [11]. Aunque ambos se basan en el aprendizaje supervisado para el entrenamiento de los algoritmos, existen ciertas diferencias en las acciones llevadas a cabo y los resultados obtenidos.

- **Clasificación:** Estos algoritmos pueden abordar tareas de: clasificación binaria, donde los datos de entrada se clasifican únicamente en dos categorías, por ejemplo: alimentando el algoritmo con datos de entrada de resultados de pruebas médicas, nos dará como salida “enfermo” o “sano”; clasificación multiclase, utilizada cuando hay más de dos categorías posibles (por ejemplo, un algoritmo que determine el tipo de buque (pesquero, carguero, de pasajeros, etc.) a partir de sus datos cinemáticos) y, por último, clasificación de etiquetas múltiples en la que un solo conjunto de datos de entrada puede estar asociado con más de una categoría simultáneamente [12].
- **Regresión:** A diferencia de la clasificación, los algoritmos de regresión están diseñados para predecir un valor continuo en lugar de categorías discretas. Su objetivo es identificar la relación entre las variables de entrada y la variable de salida, de modo que puedan realizar predicciones precisas de valores reales. Un ejemplo típico es un modelo que prediga la nota de corte de una oposición basándose en las notas de selectividad de un año específico. En este caso, los datos de

salida no son categorías, sino valores continuos, y la precisión del modelo se evalúa según la desviación entre los valores previstos y los valores reales.

Para ejecutar y resolver un problema mediante el aprendizaje automático supervisado se deben seguir los siguientes pasos que la Figura 2-1 describe gráficamente de manera sencilla y visual:

1. Selección de los tipos de datos de entrenamiento: determinar la naturaleza de los datos usados durante el entrenamiento.
2. Recopilación y limpieza de los datos de entrenamiento: asegurar la exactitud de los datos eliminando aquellos no válidos para el algoritmo.
3. Selección del algoritmo de aprendizaje supervisado a utilizar en el modelo según la naturaleza de los datos de entrada y del uso buscado.
4. Entrenamiento del modelo: ejecutar el algoritmo repetidas veces con el objetivo de mejorar la precisión y velocidad de predicción.
5. Realización de predicciones y evaluación del modelo: una vez esté suficientemente ajustado, se prueba el modelo y se valora la precisión.

Este proceso se da por terminado una vez el modelo funcione. No obstante, se debe entrenar de manera constante debido a la degradación de los datos. Esta se evita entrenando al modelo periódicamente con datos actualizados, garantizando así la precisión a lo largo del tiempo [11].

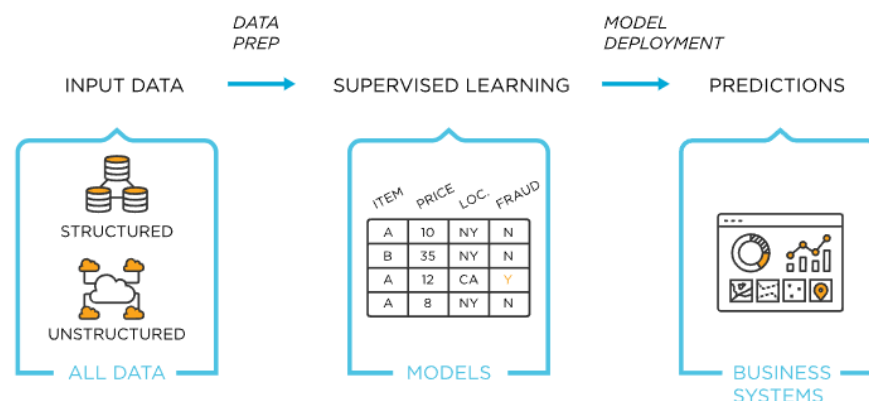


Figura 2-1 Esquema de aprendizaje supervisado [11]

2.2.2 Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, el aprendizaje no supervisado no requiere que los llamados datos de entrenamiento estén clasificados o etiquetados, aspecto de importancia fundamental en el modelo de aprendizaje anterior. Los propios modelos sin supervisión son aquellos que encuentran patrones subyacentes en los datos a analizar que, a primera vista, son imperceptibles para el ser humano. Este proceso se asemeja al proceso que ocurre en el cerebro humano cuando aprende cosas nuevas.

El aprendizaje no supervisado toma como definición según el Inesdi Business Techschool la siguiente: “El aprendizaje no supervisado es un tipo de aprendizaje automático o machine learning en el que los modelos aprenden a partir de conjuntos de datos sin etiquetar sobre el que se les permite actuar sin supervisión” [13].

Su objetivo difiere del objetivo del modelo anterior. Este no se puede aplicar a problemas de clasificación o regresión ya que el modelo de aprendizaje no supervisado carece de datos de salida. Su función consiste en encontrar la estructura oculta del conjunto de datos, agruparlos según coincidencias y devolver una representación del conjunto. En resumen, no existen resultados, no hay una forma correcta o incorrecta, simplemente existe un análisis puro de los datos.

Para comprender mejor el funcionamiento del aprendizaje no supervisado se incluye la Figura 2-2, que ilustra gráficamente su proceso interno. Como se observa en dicha figura, el proceso comienza con los datos de entrenamiento (por ejemplo, textos, documentos o imágenes), a partir de los cuales se generan vectores de características. El algoritmo se entrena utilizando estos vectores, identificando patrones y coincidencias en los datos de entrada. Una vez ajustado, el algoritmo produce un modelo funcional.

Este modelo predictivo, al recibir nuevos datos de entrada, extrae sus características y los clasifica en uno de los grupos formados previamente, basándose en las similitudes identificadas durante el entrenamiento. De esta manera, el aprendizaje no supervisado permite descubrir relaciones y estructuras ocultas en los datos sin la necesidad de etiquetas explícitas [14].

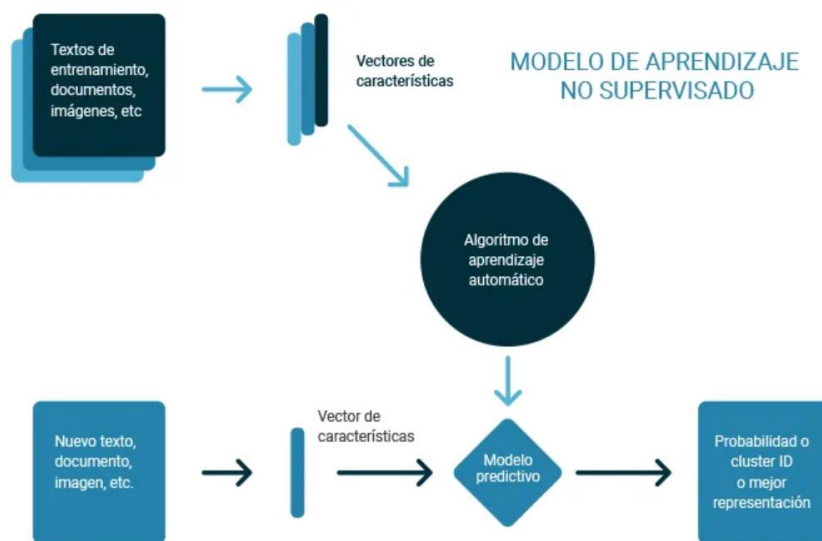


Figura 2-2 Esquema de aprendizaje no supervisado [14]

En la Figura 2-3, se muestra cómo clasificaría un modelo de aprendizaje no supervisado. Primeramente, se observa que los datos de entrada son diferentes frutas, a partir de las cuales, el modelo extraerá unas características comunes que usará para entrenarse y ser capaz de clasificar el tipo de fruta según estas similitudes que ha encontrado. Una vez el modelo está ajustado, se usan otros datos de entrada (conjunto de prueba) para así comprobar la precisión del modelo y como resultado, el modelo predictivo, agrupa estos datos de entrada según los vectores de características obtenidos de manera satisfactoria.

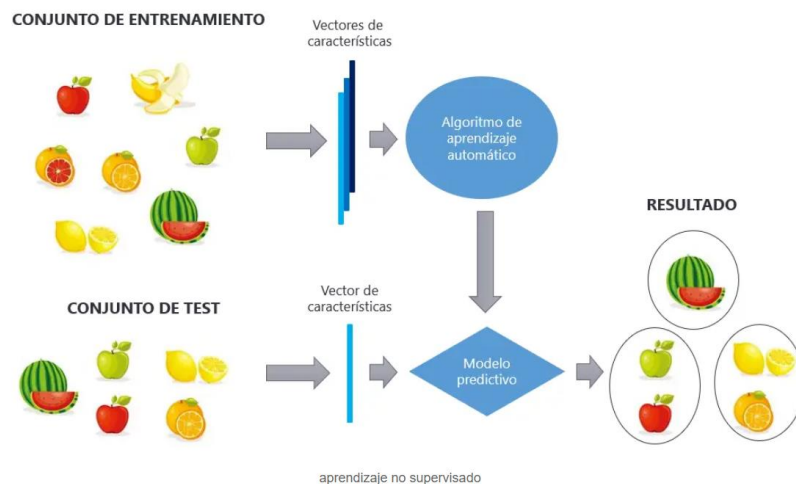


Figura 2-3 Ejemplo de funcionamiento de aprendizaje no supervisado [15]

Los algoritmos de aprendizaje no supervisado son muy variados y dependen del funcionamiento y del objetivo que se busque. Los más usados son: *K-means*, *Gaussian mixtures*, *Hierarchical clustering* y *Self-organizing maps*. A pesar de que los beneficios de los algoritmos de aprendizaje no supervisados son evidentes, el aprendizaje no supervisado presenta varios desafíos, como su elevada complejidad computacional, tiempos de entrenamiento prolongados y el riesgo de generar resultados imprecisos. Además, en ocasiones requiere la intervención humana para validar las variables de salida, y los criterios utilizados para agrupar los datos pueden carecer de claridad y transparencia. Por estas razones, en el marco de esta investigación, se ha optado por emplear un enfoque basado en técnicas de aprendizaje supervisado.

2.2.3 Aprendizaje semisupervisado

Tal como se ha visto en los apartados 2.2.1 y 2.2.2, los algoritmos de aprendizaje supervisado usan datos etiquetados para el entrenamiento del modelo, mientras que los algoritmos de aprendizaje no supervisado se entrenan mediante datos sin etiquetar. El aprendizaje semisupervisado se posiciona como un punto intermedio entre el aprendizaje supervisado y el no supervisado. Durante el proceso de entrenamiento, emplea un conjunto reducido de datos etiquetados para orientar la clasificación y la extracción de características en un conjunto de datos más amplio y no etiquetado. Esta metodología resulta especialmente útil cuando no se dispone de suficientes datos etiquetados para un enfoque supervisado o cuando el etiquetado de datos implica un coste elevado.

El aprendizaje semisupervisado surge como una respuesta para abordar las limitaciones tanto del aprendizaje supervisado como del no supervisado. Mientras que el aprendizaje supervisado destaca por su alta precisión, su implementación implica un coste computacional significativo. Esto se debe, principalmente, al proceso de etiquetado de datos, que requiere la participación de profesionales especializados en el dominio, además de enfrentar restricciones en términos de tiempo y recursos financieros.

Por su parte, el aprendizaje no supervisado, aunque menos preciso, es una metodología más sencilla y con menores costes computacionales y temporales, dado que opera con datos no etiquetados. En este contexto, el aprendizaje semisupervisado busca combinar lo mejor de ambos enfoques. Se aprovecha de la precisión de los algoritmos supervisados mientras optimiza los costes computacionales mediante el uso de datos no etiquetados. Es particularmente útil en aquellas aplicaciones donde recopilar datos etiquetados es complicado, pero, a su vez, se disponen de muchos datos no etiquetados. Se ha demostrado que el aprendizaje semisupervisado mejora mucho la precisión de los modelos en aplicaciones de reconocimiento de voz, visión por ordenador y procesamiento de lenguaje natural [16].

De forma general, el funcionamiento de los algoritmos semisupervisados se basa en el proceso ilustrado en la Figura 2-4. Este comienza con un conjunto de datos que incluye una pequeña proporción de datos etiquetados y una mayor cantidad de datos no etiquetados. Inicialmente, los datos etiquetados se emplean para entrenar un modelo utilizando técnicas supervisadas. Una vez entrenado, este modelo se utiliza para predecir las etiquetas de los datos no etiquetados. Posteriormente, los nuevos datos etiquetados, junto con los etiquetados inicialmente, se combinan para formar un conjunto de datos ampliado, que se emplea para entrenar un modelo final mediante aprendizaje supervisado. Este enfoque permite mejorar la precisión y la eficiencia del modelo al tiempo que reduce los costes asociados al etiquetado masivo de datos.

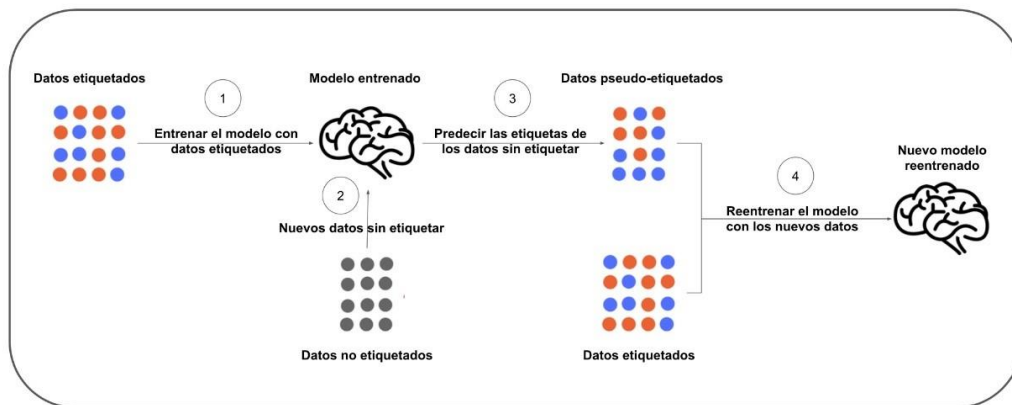


Figura 2-4 Estructura de funcionamiento de aprendizaje semisupervisado [12]

2.2.4 Aprendizaje por refuerzo

El aprendizaje por refuerzo es una técnica de ML diseñada para entrenar modelos capaces de tomar decisiones óptimas en función de su interacción con un entorno. Más que decisiones, genera estrategias automáticamente. Este enfoque imita el proceso de aprendizaje por ensayo y error propio de los seres humanos, donde las acciones que conducen al logro de un objetivo se refuerzan, mientras que aquellas que se desvían del objetivo son descartadas.

Los algoritmos de *Reinforcement Learning* (RL) se basan en un esquema de recompensa y castigo para procesar los datos y ajustar su comportamiento. Aprenden de la retroalimentación obtenida tras cada acción, identificando de manera autónoma las rutas óptimas para alcanzar un resultado deseado. Una característica distintiva del RL es su capacidad para gestionar gratificación aplazada: las estrategias más efectivas pueden implicar sacrificios a corto plazo en favor de recompensas más significativas a largo plazo. Esto permite que el modelo explore diferentes opciones y ajuste su política de decisiones en consecuencia, incluso si inicialmente implica penalizaciones [17].

Para proporcionar una definición más precisa del aprendizaje por refuerzo y describir su funcionamiento, es fundamental primero definir los componentes y la relaciones entre sí que se utilizan con frecuencia en esta área del ML [18]:

- **Agente:** Es el programa o modelo que se entrena y que posteriormente será responsable de tomar decisiones para llevar a cabo la tarea requerida.
- **Entorno:** Es el espacio, ya sea real o virtual, en el que opera el agente. Representa el contexto en el que se desarrollan las interacciones del sistema.
- **Acción:** Son las decisiones o movimientos realizados por el agente dentro del entorno, que generan un cambio en el estado de este.
- **Estado:** Es la condición actual del entorno que resulta de las acciones realizadas por el agente, reflejando cómo ha cambiado el entorno tras dichas interacciones.
- **Recompensas:** a partir de cada acción que el agente toma, se obtendrá un premio o una penalización que ayudará al agente a entender si lo está haciendo bien o mal.

En la Figura 2-5 se representan gráficamente los componentes del aprendizaje por refuerzo y las relaciones que mantienen entre sí para el funcionamiento del algoritmo. Primeramente, el agente recibe el estado inicial del ambiente y toma una acción que influye en el ambiente. Esta decisión tomada por el agente tendrá unas repercusiones: en la siguiente iteración, el agente recibirá el nuevo estado y la recompensa (positiva o negativa) obtenida. Una vez se reciba la recompensa, existen dos opciones: si la recompensa es positiva, se refuerza el comportamiento del agente para el futuro; en cambio, si la

recompensa es negativa, se le transmite al agente que, ante la misma situación, debe actuar de manera diferente.

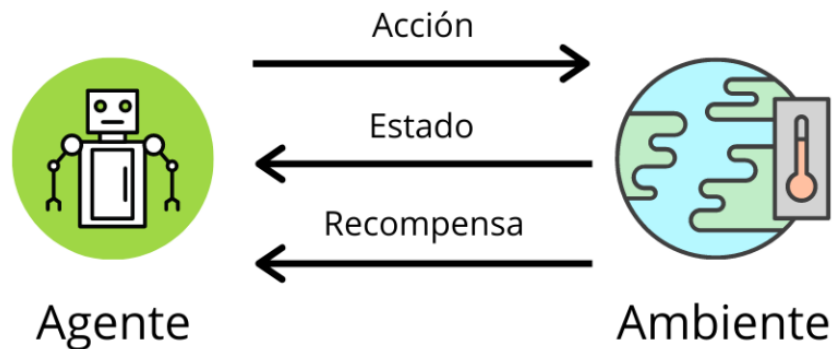


Figura 2-5 Representación de los componentes y relaciones entre sí en el aprendizaje por refuerzo [18]

El aprendizaje por refuerzo se basa en el proceso de decisión de Markov. Los Procesos de Decisión de Markov (MDP, *Markov Decision Processes*) representan una formulación matemática idealizada para abordar el problema del aprendizaje por refuerzo. Los MDP proporcionan un enunciado teórico preciso que describe formalmente el entorno en el que opera el aprendizaje por refuerzo, asumiendo que dicho entorno es completamente observable. Esta característica permite que muchos de los problemas dentro del aprendizaje por refuerzo puedan ser formalizados como MDP. Junto al MDP se usan varios elementos claves para la resolución matemática del problema, como el retorno, las funciones de valor y las ecuaciones de Bellman [19].

Volviendo al aprendizaje por refuerzo, este se presenta como una herramienta poderosa dentro del ámbito de la inteligencia artificial y tiene muchos beneficios. Entre todos destacamos los siguientes: permite a los sistemas operar de manera eficiente en entornos dinámicos y desconocidos, descubriendo soluciones óptimas que serían difíciles de diseñar mediante métodos tradicionales; requiere menos interacción humana siendo capaz de aprender por sí solo a diferencia de los algoritmos de ML tradicionales que precisan de acciones humanas para etiquetar pares de datos necesarios para entrenar al algoritmo. Por último, es óptimo con objetivos largoplacistas consiguiendo que el aprendizaje por refuerzo sea apto para situaciones en las que las acciones tomadas tienen consecuencias prolongadas en el tiempo [17].

2.2.5 Deep Learning

El *Deep Learning* es un subcampo del ML que, a su vez, está englobado dentro de la inteligencia artificial, que se especializa en el desarrollo de modelos basados en redes neuronales profundas tratando de simular el complejo poder de toma de decisión del cerebro humano. Estas redes están diseñadas para procesar grandes volúmenes de datos complejos y generar decisiones altamente precisas en tareas específicas. Su aplicación es particularmente efectiva en contextos donde la cantidad y la complejidad de los datos dificultan el uso de enfoques tradicionales [20]. En la Figura 2-6 se visualiza la relación entre los subconjuntos de inteligencia artificial.

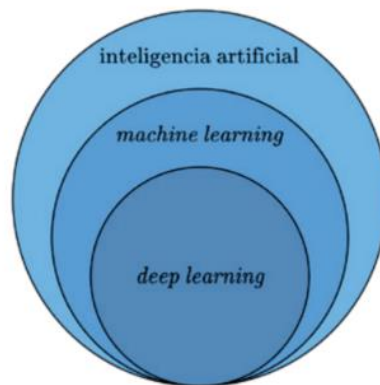


Figura 2-6 Relación entre la inteligencia artificial, ML y *Deep Learning* [21]

La principal diferencia entre el *Deep Learning* y el ML radica en la arquitectura de las redes neuronales subyacentes. Los modelos tradicionales de ML, denominados "no profundos", utilizan redes neuronales simples con una o dos capas computacionales. En contraste, los modelos de *Deep Learning* emplean redes más complejas con tres o más capas, llegando incluso a cientos o miles de capas para entrenar los modelos. En la Figura 2-7 se muestra una comparativa entre el desarrollo de un sistema basado en el ML y *Deep Learning*. La principal diferencia reside en que las técnicas de *Deep Learning* se encargan de extraer las características más importantes de cara a la resolución del problema, además de aprender la transformación que deben sufrir estas características para obtener la salida deseada.

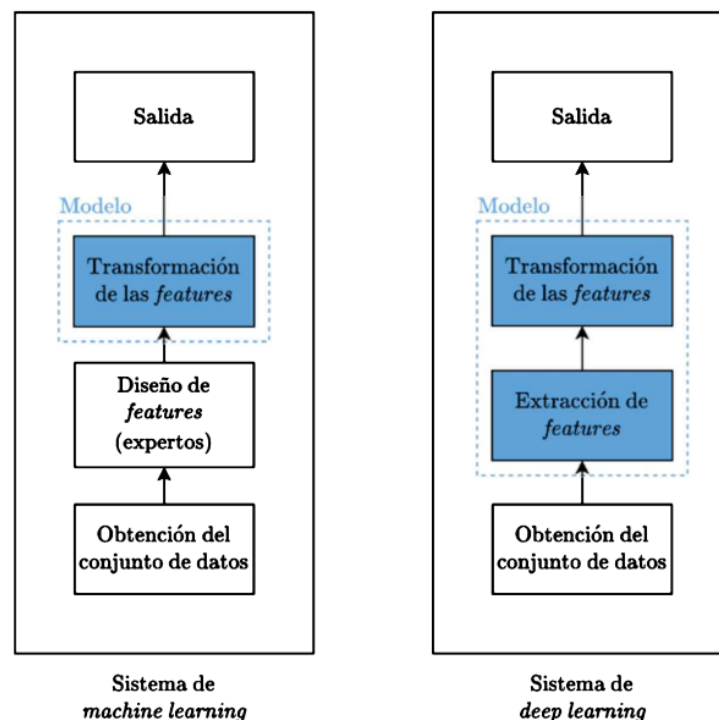


Figura 2-7 Comparación del funcionamiento de sistemas de ML y DL [21]

Mientras que los modelos de aprendizaje supervisado dependen de datos estructurados y etiquetados para generar predicciones precisas, los modelos de *Deep Learning* pueden funcionar con aprendizaje no supervisado. Este enfoque permite a las redes de *Deep Learning* identificar características, patrones y relaciones directamente a partir de datos en bruto y no estructurados. Además, tienen la capacidad de evaluar y ajustar sus propios resultados para mejorar la precisión de las predicciones.

El *Deep Learning* es un componente fundamental de la ciencia de datos que impulsa una amplia gama de aplicaciones y servicios orientados a la automatización. Estos sistemas pueden llevar a cabo tareas tanto analíticas como físicas sin intervención humana [22].

El funcionamiento de un modelo de *Deep Learning* puede dividirse en dos etapas principales. La primera es la extracción de características, en la cual el modelo analiza los datos de entrada para generar representaciones relevantes y organizarlas en una jerarquía de complejidad creciente. La segunda etapa, la transformación de características, utiliza las representaciones extraídas para producir la salida esperada del modelo. Por ejemplo, en una red neuronal convolucional diseñada para clasificar imágenes, las capas iniciales detectan bordes, las intermedias identifican formas y texturas, y las finales predicen la clase a la que pertenece la imagen procesada.

En los primeros años del desarrollo de redes neuronales profundas, el entrenamiento de estos modelos enfrentaba grandes limitaciones debido a la falta de capacidad computacional y a la dificultad de ajustar los parámetros de redes complejas. Este panorama cambió en 2006 cuando Geoffrey Hinton y sus colaboradores introdujeron métodos que facilitaron el entrenamiento de redes más profundas, marcando un hito en el campo. Posteriormente, en 2012, el modelo *AlexNet* revolucionó el *Deep Learning* al reducir significativamente el error en la clasificación de imágenes durante la competición *ImageNet*, utilizando redes neuronales convolucionales y aprovechando el paralelismo computacional de las GPU para entrenar redes complejas en tiempos relativamente cortos.

Sin embargo, el *Deep Learning* también presenta desafíos significativos. En escenarios con pocos datos, los modelos tienden a sobreajustarse, lo que dificulta su capacidad para generalizar. Asimismo, el entrenamiento de redes neuronales profundas implica un alto coste computacional y de almacenamiento, lo que puede limitar su aplicabilidad en ciertos contextos.

A pesar de estas limitaciones, los avances tecnológicos en el procesamiento de datos, el almacenamiento y la computación han impulsado el auge del *Deep Learning*. Este enfoque ha transformado el campo de la inteligencia artificial, permitiendo su aplicación en áreas como la conducción autónoma, la detección temprana de enfermedades y el desarrollo de sistemas avanzados de inteligencia artificial o habilitando tecnologías de uso cotidiano como asistentes digitales, controles remotos de TV activados por voz, detección de fraudes con tarjetas de crédito, vehículos autónomos [21].

2.2.6 Campos de aplicación

El aprendizaje automático ha supuesto un gran avance tecnológico debido al gran abanico de aplicaciones que tiene. A continuación, se expondrán algunos de estos campos, enfocándose en los relativos a la vida cotidiana [23].

2.2.6.1 Medicina

Los avances en el ML han permitido entrenar a las máquinas para el reconocimiento de patrones, una técnica que se emplea cada vez más en la obtención y análisis de imágenes radiológicas. La visión artificial, habilitada por esta tecnología, se utiliza frecuentemente para el análisis de mamografías y la detección temprana del cáncer de pulmón. Se estima que los médicos que evalúan las mamografías para identificar el cáncer de mama pasan por alto hasta el 40% de los casos, pero el ML tiene el potencial de mejorar significativamente esta tasa. Además, el aprendizaje automático se emplea para clasificar tumores, detectar fracturas óseas que son difíciles de identificar a simple vista y diagnosticar trastornos neurológicos.

Asimismo, el ML se aplica en el análisis de los registros médicos históricos de los pacientes y sus resultados, con el objetivo de desarrollar nuevos planes de tratamiento. En el ámbito de la investigación genética, la modificación genética y la secuenciación del genoma, esta tecnología ayuda a identificar cómo los genes influyen en la salud. El ML puede detectar marcadores genéticos y genes que responden o no a tratamientos o medicamentos específicos, así como aquellos que pueden provocar efectos

secundarios importantes en ciertas personas. Estos análisis avanzados permiten generar recomendaciones personalizadas de medicamentos o tratamientos basadas en datos precisos y específicos de cada paciente.

2.2.6.2 Filtrado de correo electrónico

Los algoritmos de ML en *Gmail* de *Google* automatizan el proceso de filtrado de correos electrónicos, clasificándolos en categorías como principal, social y promociones, mientras que también detectan y redirigen el correo no deseado hacia la carpeta de spam. Más allá de las reglas tradicionales de las aplicaciones de correo, las herramientas de ML también pueden optimizar la gestión del correo electrónico, clasificándolo para que sea enviado a las personas correspondientes para una acción más rápida, moviendo archivos adjuntos a las ubicaciones apropiadas, entre otras funciones. Por ejemplo, herramientas de automatización en la gestión de correos electrónicos, como *Levity*, utilizan algoritmos de clasificación de texto basados en ML para identificar y categorizar los correos electrónicos a medida que llegan. Esto permite generar respuestas personalizadas según la categoría del correo, lo que ahorra tiempo y puede contribuir a mejorar la tasa de conversión.

2.2.6.3 Marketing y ventas

Los algoritmos de ML y la ciencia de datos juegan un papel clave en los motores de recomendación de plataformas como *Amazon*, *Netflix* y *StitchFix*, que sugieren productos o contenidos a los usuarios en función de sus preferencias, patrones de navegación e historial de compras. Además, el ML facilita la creación de campañas de marketing personalizadas al identificar las ofertas más relevantes para cada cliente. A partir de esta información, se pueden ajustar los materiales promocionales para alinearse con los intereses específicos de cada usuario. También permite realizar un seguimiento preciso de las campañas mediante el análisis de métricas como las tasas de apertura y clics, lo que ayuda a optimizar la efectividad de las estrategias.

2.2.6.4 Asistentes personales y de voz

El aprendizaje automático es la tecnología subyacente que impulsa las funciones de los asistentes virtuales y de voz, como *Alexa* de *Amazon* y *Siri* de *Apple*. Esta interacción generalmente involucra tareas como el reconocimiento de voz, la conversión de voz a texto, el procesamiento de lenguaje natural y la conversión de texto a voz. Cuando un usuario formula una pregunta a un asistente virtual, el ML es responsable de buscar una respuesta o de hacer referencia a preguntas previas realizadas por el usuario, facilitando la interacción.

Asimismo, el ML es esencial en el funcionamiento de los *bots* de mensajería, como aquellos integrados en plataformas como *Facebook Messenger* y *Slack*. En el caso de *Facebook Messenger*, el ML optimiza los *chatbots* de atención al cliente, permitiendo a las empresas ofrecer respuestas rápidas, mostrar carruseles de imágenes y botones interactivos, ayudar a los usuarios a localizar opciones cercanas, realizar un seguimiento de los envíos y facilitar compras seguras. Además, Facebook emplea esta tecnología para monitorizar los chats en Messenger, detectando fraudes o interacciones no deseadas, como la actividad sospechosa de adultos que envían múltiples solicitudes de amistad o mensajes a menores de 18 años.

2.3 Algoritmos de aprendizaje automático supervisado

Los algoritmos de aprendizaje automático supervisado son fundamentales en la predicción del tipo de buque, ya que permiten clasificar los datos en función de patrones previamente aprendidos. Habiendo expuesto los diferentes modelos de aprendizaje del ML, se presentan los algoritmos más relevantes para este trabajo, destacando sus características principales y su relación con la mejora de la eficiencia computacional: Árbol de decisión, *Random Forest* y *XGBoost*.

2.3.1 Árboles de decisión

Los árboles de decisión son una técnica de aprendizaje supervisado aplicable tanto a problemas de clasificación como de regresión. No obstante, su uso más común se da en tareas de clasificación. Este enfoque se asemeja a un clasificador estructurado como un diagrama de flujo, que guía el proceso de toma de decisiones de manera jerárquica y secuencial [24]. El árbol de decisión está formado por nodos de distintos niveles y ramas que los unen [25]:

- **Nodo raíz:** es el primer nodo del árbol de decisión, no tiene ninguna rama entrante. Contiene el conjunto de datos al completo que, a medida que se avance en el árbol de decisión, se irá dividiendo en subconjuntos. En él, se produce la primera división que depende de la variable más importante o decisiva.
- **Nodo interno o de decisión:** los encontramos tras la primera división y vuelven a dividir el conjunto de datos según otra variable diferente a la primera.
- **Nodo hoja:** es el nodo final, se encuentra en la parte más inferior del esquema e indica la clasificación definitiva sin generar más divisiones.
- **Rama:** representa la decisión en función de la variable tomada.

En la Figura 2-8 se muestra un ejemplo de árbol de decisión en el que se distinguen las partes mencionadas anteriormente mediante un código de colores. El nodo raíz es el recuadro amarillo en el que se recogen todos los datos sobre la marca de los automóviles en este caso. Además, tiene tres nodos de decisión (recuadro azul) en los que se subdividen los datos en base a las variables escritas en ellos. Por último, con círculos rojos y verdes se identifican los nodos hojas, obteniendo como resultado comprar o no comprar el automóvil basándose en las decisiones tomadas anteriormente.

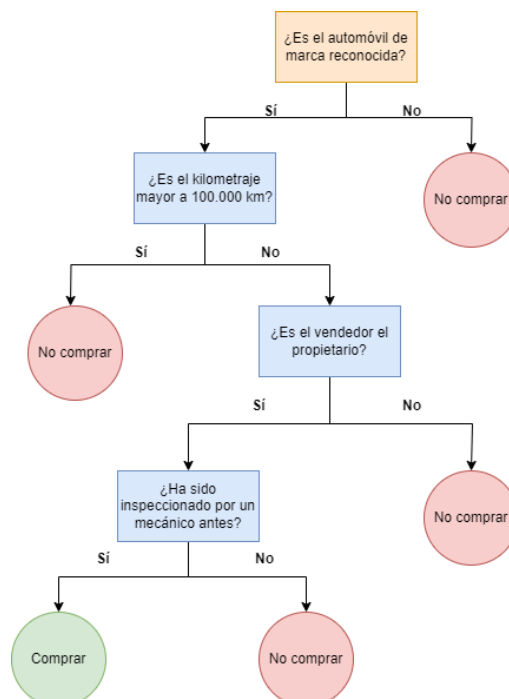


Figura 2-8 Ejemplo de árbol de decisión [26]

El aprendizaje basado en árboles de decisión se fundamenta en un enfoque de divide y vencerás, mediante una búsqueda iterativa destinada a identificar los puntos de división óptimos en la estructura del árbol. Este proceso, ejecutado de forma descendente y recursiva, continúa hasta clasificar la mayor parte de los datos en grupos asociados a etiquetas de clase específicas. La capacidad de un árbol de decisión para clasificar los datos en conjuntos homogéneos depende en gran medida de su nivel de complejidad. En árboles más simples, los nodos terminales tienden a ser más puros, es decir, contienen datos de una sola clase. Sin embargo, al incrementar la complejidad del árbol, mantener esta pureza se

torna más difícil, lo que puede dar lugar a la fragmentación de los datos y, en última instancia, al sobreajuste del modelo.

Por este motivo, se priorizan árboles de decisión más sencillos, en consonancia con el principio de parsimonia de la Navaja de Occam, que establece que no se debe añadir complejidad innecesaria. Para controlar este riesgo y optimizar el modelo, se recurre a la poda, un procedimiento que elimina las ramas relacionadas con características de menor relevancia. Posteriormente, la evaluación del modelo puede realizarse mediante técnicas como la validación cruzada, garantizando así su capacidad de generalización y ajuste a nuevos datos.

Adicionalmente, los árboles de decisión pueden mejorar su precisión al ser integrados en métodos de ensamblaje, como el algoritmo de bosque aleatorio. Este enfoque combina múltiples árboles de decisión que no están correlacionados entre sí, permitiendo obtener predicciones más precisas y robustas, especialmente en escenarios donde los árboles individuales presentan limitaciones [24].

2.3.2 *Random Forest*

Los árboles de decisión presentan algunas limitaciones que los hacen muy efectivos para los datos de entrenamiento, pero cuando se aplican a datos nuevos, su precisión suele disminuir. Esto ocurre porque tienden a sobreajustarse al conjunto de entrenamiento (*overfitting*). Para mitigar estos problemas, surge *Random Forest*, un método que mejora la precisión y reduce el sobreajuste [27]. El sobreajuste se produce cuando el modelo brinda predicciones precisas para datos de entrenamiento, pero no para datos nuevos. Esto ocurre cuando el modelo se ajusta demasiado al conjunto de datos de entrenamiento [28].

Random Forest es una técnica de aprendizaje supervisado empleada para la resolución de problemas de clasificación y de regresión y fue registrado por Leo Breiman y Adele Cutler. *Random Forest* pertenece a la familia de algoritmos conocidos como métodos de ensamble, lo que significa una combinación de modelos individuales simples para crear en conjunto uno más potente. Para entender el funcionamiento de *Random Forest* es obligatorio conocer cómo trabajan los árboles de decisión, explicado en el apartado 2.3.1.

El algoritmo *Random Forest* se fundamenta en la generación de un conjunto de n árboles de decisión no podados, los cuales se construyen a partir de subconjuntos de datos de entrenamiento seleccionados aleatoriamente a partir del conjunto total disponible. Estos subconjuntos, conocidos como *Bootstrap Datasets*, se obtienen mediante muestreo con reemplazo, lo que implica que pueden contener observaciones repetidas y excluir ciertas instancias del *dataset* original. Este enfoque, denominado *bagging* (*Bootstrap Aggregating*), contribuye a la reducción de la varianza del modelo y mejora su capacidad de generalización.

Durante la fase de entrenamiento, el usuario define las características a considerar en la construcción de los árboles, lo que garantiza que cada uno de ellos siga una estructura diferente y mantenga cierto grado de independencia con respecto a los demás, siendo esta una de las características más importantes de *Random Forest*. Paralelamente, se reserva un subconjunto de datos del *dataset* original que no es incluido en ningún *Bootstrap Dataset*. Este conjunto, denominado datos de prueba (*test set*), se emplea posteriormente para evaluar el desempeño del modelo y estimar su capacidad de generalización.

Una vez entrenado el conjunto de árboles de decisión, la inferencia del modelo se lleva a cabo de manera sencilla, tanto para problemas de clasificación como de regresión. Ante la llegada de una nueva instancia, esta se introduce en cada uno de los árboles generados durante la fase de entrenamiento. En el caso de una tarea de clasificación, cada árbol emite una predicción individual, y el resultado final se obtiene mediante votación mayoritaria entre las predicciones generadas. Para problemas de regresión, la salida del modelo corresponde al valor promedio de las predicciones proporcionadas por los árboles individuales. Este proceso de agregación de resultados permite que *Random Forest* presente una mayor robustez frente al sobreajuste en comparación con un único árbol de decisión [29].

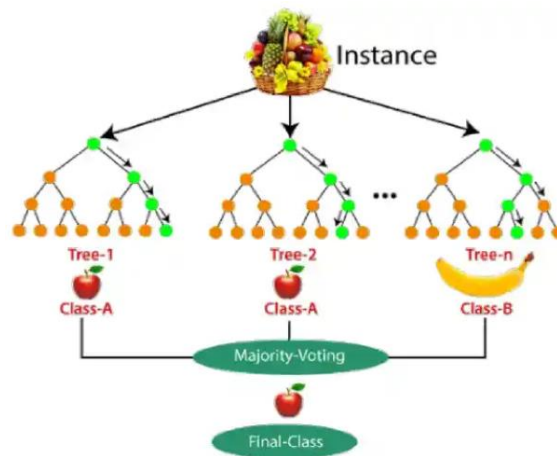


Figura 2-9 Ejemplo de predicción mediante *Random Forest* [30]

Se ejemplifica el funcionamiento de *Random Forest* en la Figura 2-9. Consideremos una canasta de frutas como el conjunto de datos. A partir de esta canasta, se extraen n muestras sobre las cuales se construye un árbol de decisión individual para cada una. Cada uno de estos árboles genera una predicción independiente. El resultado final del modelo se determina mediante un esquema de votación mayoritaria. En la Figura 2-9, se observa que la mayoría de los árboles de decisión predicen una manzana en comparación con una banana. Dado que la opción predominante es la manzana, esta se considera como la predicción final del modelo.

A pesar de que *Random Forest* está compuesto de árboles de decisión, existen algunas diferencias. *Random Forest* tiene la ventaja frente al árbol de decisión de que es capaz de evitar el sobreajuste que limita al árbol de decisión, además de ofrecer mayor precisión y capacidad de trabajar con volúmenes de datos mayores. En cambio, el árbol de decisión requiere menor coste computacional y tiempo de ejecución. En la Tabla 2-2 se exponen algunas de las diferencias entre estos dos algoritmos.

Árboles de decisión	<i>Random Forest</i>
1. Los árboles de decisión normalmente sufren el sobreajuste si no se controla su crecimiento.	1. <i>Random Forest</i> se crea a partir de subconjuntos de datos y el resultado final se basa en una clasificación por promedio o por mayoría de los resultados independientes proporcionados por los árboles de decisión, solucionando así el problema de sobreajuste.
2. Un solo árbol de decisión es más rápido computacionalmente hablando.	2. Es relativamente más lento en comparación con el árbol de decisión.
3. Cuando un árbol de decisión recibe un conjunto de datos con características como entrada, generará un conjunto de reglas para realizar predicciones.	3. Selecciona aleatoriamente observaciones, construye múltiples árboles de decisión y obtiene el resultado final mediante un promedio en caso de regresión o votación mayoritaria en clasificación. No se basa en un conjunto fijo de fórmulas.

Tabla 2-2 Comparación entre *Random Forest* y árbol de decisión [30]

2.3.3 XGBoost

XGBoost (eXtreme Gradient Boosting) [31] fue desarrollado por Tianqi, en la Universidad de Washington. *XGBoost* es un algoritmo de aprendizaje automático supervisado de clasificación y regresión y de código abierto que está basado en árboles de decisión y desarrolla una mejora sobre otros métodos comentados en los apartados 2.3.1 y 2.3.2.

XGBoost se basa en un ensamble al igual que *Random Forest*. Usa árboles de decisión, característica que comparte con *Random Forest*, pero se diferencia en el proceso de construcción de los árboles de decisión y en la interpretación de los resultados. Los bosques aleatorios se basan en el concepto de *bagging* (agregación *bootstrap*) y entrenan cada árbol de forma independiente para combinar sus predicciones, mientras que los algoritmos de *boosting* utilizan un enfoque aditivo en el que los aprendices débiles se entrenan secuencialmente para corregir los errores de los modelos anteriores [32].

Los árboles de decisión potenciados por gradientes representan una implementación avanzada de algoritmos de *boosting*, los cuales emplean el descenso de gradiente como técnica de optimización. Similar a otras metodologías de *boosting*, este enfoque comienza con un aprendiz débil que realiza las predicciones iniciales, siendo este aprendiz base generalmente un árbol de decisión. Los árboles subsecuentes se generan de manera aditiva, ajustándose a los errores cometidos por el aprendiz base en las iteraciones previas.

El algoritmo evalúa el desempeño de cada árbol a través del cálculo de los residuos, definidos como la diferencia entre las predicciones del modelo y los valores reales. Estos residuos se utilizan para actualizar el modelo iterativamente mediante la minimización de una función de pérdida, lo que permite mejorar la precisión de las predicciones.

En el contexto de ML, las funciones de pérdida son fundamentales para evaluar la calidad de un modelo. En los árboles de decisión potenciados por gradientes, el término "gradiente" se refiere al proceso de descenso de gradiente, una técnica de optimización ampliamente utilizada para reducir la función de pérdida y, en consecuencia, mejorar el rendimiento del modelo durante el entrenamiento.

El descenso de gradiente ajusta los parámetros del modelo minimizando iterativamente el valor de la función de pérdida. Ejemplos comunes de funciones de pérdida incluyen el error cuadrático medio (MSE) y el error absoluto medio (MAE) en problemas de regresión, así como la pérdida de entropía cruzada para problemas de clasificación. Además, es posible desarrollar funciones de pérdida personalizadas que se adapten a casos de uso específicos y características particulares del conjunto de datos.

Este enfoque iterativo y optimizado convierte a los árboles de decisión potenciados por gradientes en una herramienta poderosa dentro del aprendizaje supervisado, destacándose en tareas tanto de clasificación como de regresión [33].

2.4 Conocimiento del Entorno Marítimo

El Conocimiento del Entorno Marítimo (CEM) se refiere al proceso de recopilación, fusión y análisis de información proveniente de múltiples fuentes con el propósito de generar una visión precisa de las actividades en los espacios marítimos de interés nacional. A través de este procedimiento continuo de síntesis y evaluación de datos, se logra una representación detallada y casi en tiempo real de los eventos que ocurren en nuestras aguas. Esta representación es conocida como *Recognized Maritime Picture* (RMP) [34]. En la Figura 2-10 se muestra un ejemplo de RMP en la que no se han clasificado los contactos.



Figura 2-10 Ejemplo de una RMP sin clasificar [35]

España posee una ubicación geoestratégica privilegiada que la sitúa como un actor clave en el transporte marítimo global. Su posición entre Europa, África y América, reforzada por la presencia del archipiélago canario, le otorga un papel destacado en las rutas comerciales internacionales. Actualmente, se emplea el transporte marítimo por aproximadamente el 80% de sus importaciones, incluyendo la mayoría de los hidrocarburos, así como la mitad de sus exportaciones.

Además de su importancia en el comercio internacional, España cuenta con una de las flotas pesqueras más relevantes a nivel mundial y una considerable actividad marítima vinculada al turismo y las prácticas náuticas deportivas, sectores que desempeñan un papel esencial en la economía del país. Sin embargo, el panorama de la seguridad marítima se ha visto cada vez más amenazado, especialmente tras los atentados del 11 de septiembre de 2001. Entre los riesgos más significativos se incluyen el terrorismo internacional, la piratería, el tráfico ilícito de mercancías y personas, la pesca ilegal y la contaminación del medio marino.

Ante estos desafíos, la Armada Española ha evolucionado constantemente para garantizar la seguridad en los espacios marítimos de interés nacional. En este contexto, en 2002, bajo el gobierno de José María Aznar, se creó la Fuerza de Acción Marítima (FAM), una de las tres principales fuerzas bajo el mando del Almirante de la Flota (ALFLOT), junto con la Fuerza de Acción Naval y la Infantería de Marina [36].

La capacidad actual para obtener un conocimiento preciso del entorno marítimo se sustenta en el Centro de Operaciones y Vigilancia de Acción Marítima (COVAM), el cual actúa como la principal herramienta de la FAM para recopilar y analizar información proveniente de diversas fuentes. Esta labor resulta esencial para garantizar un CEM integral, alineándose con los objetivos establecidos en la Estrategia Nacional de Seguridad Marítima de 2024 (ENSM 2024) [37]. A través del COVAM, se fortalece la vigilancia y el control de los espacios marítimos de interés, permitiendo a las Fuerzas Armadas anticiparse a posibles amenazas y contribuir a la protección de los intereses nacionales en el ámbito marítimo. En la Figura 2-11 se observa la representación de las distintas zonas en las que se dividen las aguas de responsabilidad nacional. Es en todas ellas en las que el COVAM aplica sus funciones de vigilancia y control.

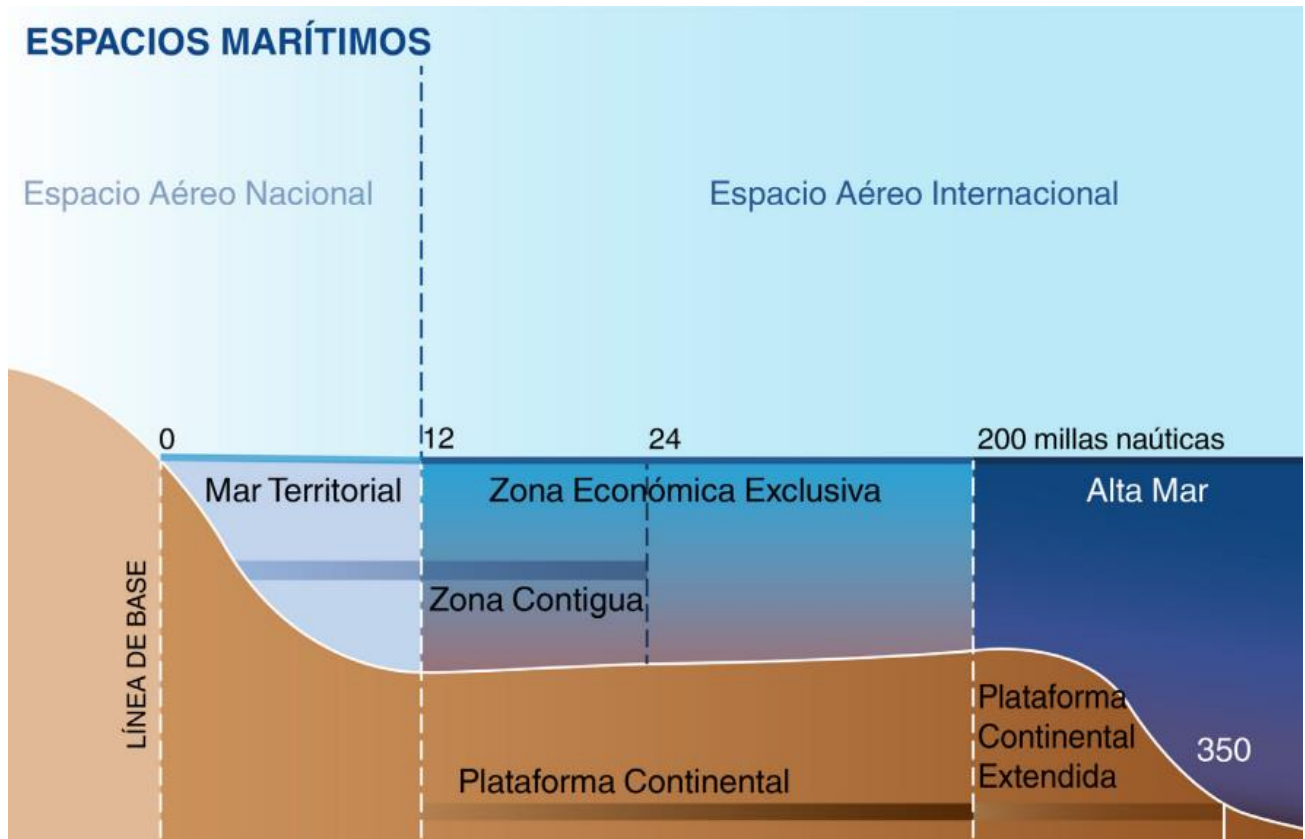


Figura 2-11 Espacios marítimos [37]

2.4.1 Estrategia Nacional de Seguridad Marítima

España, como nación marítima, depende en gran medida de la seguridad en sus aguas y en las rutas de navegación internacionales. La ENSM 2024 establece un marco integral para la protección de los intereses nacionales en el ámbito marítimo, destacando la necesidad de fortalecer las capacidades de vigilancia, control y análisis del entorno marítimo. En un contexto global donde la competencia geoestratégica se intensifica y los riesgos asociados a la seguridad marítima son cada vez más complejos, la ENSM 2024 enfatiza la importancia de la tecnología y la inteligencia artificial en la optimización de la seguridad y la gestión del dominio marítimo.

Gracias al CEM desarrollado por el COVAM, se permite la detección de patrones de tráfico marítimo, la identificación de comportamientos anómalos y la prevención de amenazas en el entorno marítimo. La inteligencia artificial y el aprendizaje automático se posicionan como herramientas clave en este proceso, al permitir un análisis más rápido y preciso de grandes volúmenes de datos, mejorando la toma de decisiones y reduciendo los tiempos de respuesta ante eventos de seguridad.

La ENSM 2024 también identifica una serie de riesgos y amenazas que afectan a la seguridad marítima, entre los que destacan las estrategias híbridas, que combinan ciberataques, sabotajes y presiones económicas para desestabilizar el dominio marítimo. Además, se señala la importancia de proteger infraestructuras críticas, como gasoductos, cables submarinos y puertos estratégicos, cuya vulnerabilidad puede tener graves consecuencias económicas y de seguridad. La piratería, el narcotráfico y otros tráficos ilícitos continúan siendo amenazas significativas, mientras que el impacto del cambio climático y la sobreexplotación de los recursos marinos añaden un nuevo nivel de complejidad a la seguridad marítima. En este contexto, la integración de sistemas avanzados de predicción y detección de anomalías mediante modelos de inteligencia artificial y procesamiento de datos puede contribuir significativamente a mitigar estos riesgos [37].

2.4.2 COVAM

El Almirante de Acción Marítima (ALMART) desempeña las funciones de Comandante del Mando Operativo Marítimo (CMOM), formando parte de la estructura operativa de las Fuerzas Armadas. Este mando tiene la responsabilidad de planificar, dirigir y supervisar las operaciones de vigilancia y seguridad en los espacios marítimos bajo responsabilidad, soberanía e interés nacional. Para cumplir con esta misión, el CMOM mantiene una estrecha coordinación con diversas autoridades y organismos, tanto civiles como militares, a nivel nacional e internacional, garantizando una respuesta eficiente y coordinada ante cualquier desafío en el ámbito marítimo.

El ALMART es el responsable del mantenimiento del Conocimiento del Entorno Marítimo en los espacios de interés nacional. Además, reside en él la responsabilidad de activar y coordinar el Sistema de Cooperación y Orientación al Tráfico Marítimo (NCAGS).

El ALMART y el CMOM, entendidos como la misma persona, pero con distintas funciones, cuenta con el COVAM para el ejercicio de sus cometidos. Este nace con vocación de apoyo a la Comunidad Marítima, no solo en aguas de soberanía española, sino también lejos de España, como el Golfo de Guinea, África Occidental, India, Asia, entre otros [38]. En la Figura 2-12 se ofrece una visión geoespacial de los espacios marítimos de interés para el COVAM fuera de las aguas nacionales.

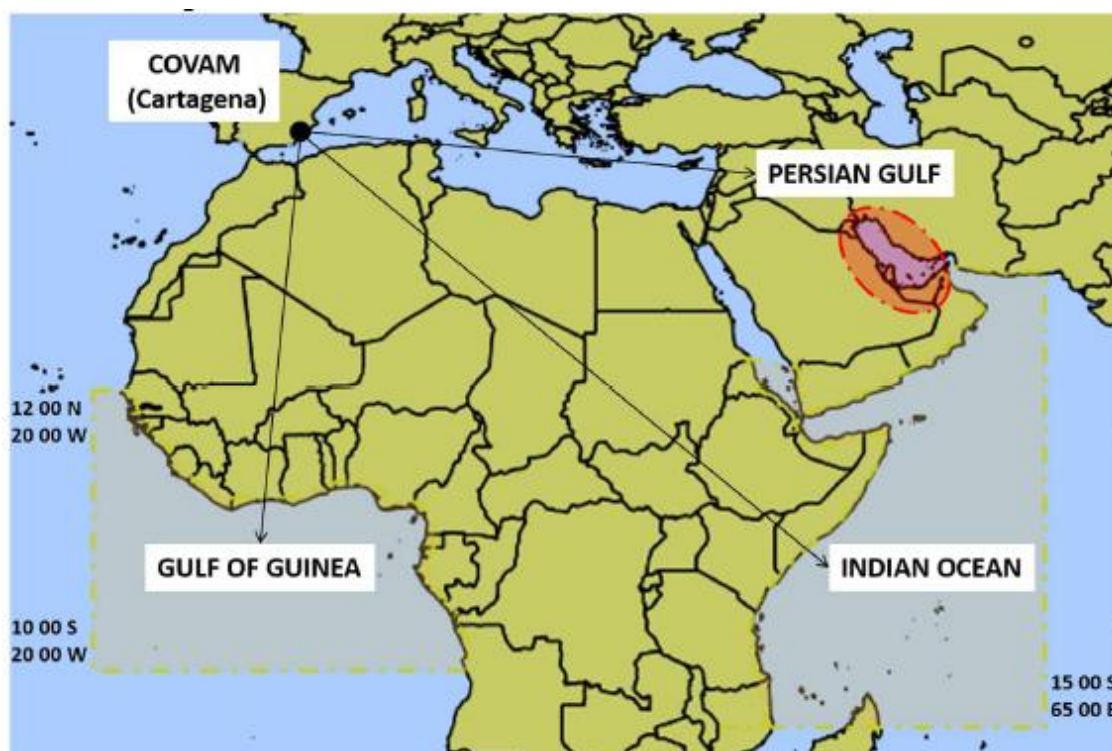


Figura 2-12 Espacios marítimos de interés [12]

El COVAM está ubicado en Cartagena y encuadrado en el Estado Mayor de la Fuerza de Acción Marítima. Aquí, se integran y analizan todos los datos relacionados con los espacios marítimos de interés, un proceso conocido como CEM o RMP. El COVAM es una herramienta esencial para la FAM en la gestión del CEM, proporcionando un servicio no solo a la Armada, sino también a otros organismos de la comunidad marítima. Esto permite mejorar la eficiencia en la ejecución de las misiones de los buques de la FAM, al mismo tiempo que facilita la cooperación y el apoyo a diversas entidades que operan en el ámbito marítimo.

En cuanto a sus funciones, el COVAM fue creado con el propósito de respaldar la ejecución de las misiones asignadas a la FAM. Desde sus instalaciones, se lleva a cabo una vigilancia marítima continua, utilizando tanto recursos tecnológicos como capacidades humanas especializadas. Esta vigilancia se traduce en operaciones de monitorización y control, cuyo objetivo es proporcionar seguridad a los

buques que transitan por las áreas de influencia de España. De esta manera, el COVAM contribuye activamente a la protección de los espacios marítimos de interés nacional y a la garantía de la seguridad marítima en un entorno cada vez más complejo y dinámico.

Para llevar a cabo sus funciones de manera eficiente y garantizar su operatividad de forma ininterrumpida, el COVAM dispone de un equipo compuesto por 35 profesionales, distribuidos estratégicamente para cubrir los puestos esenciales en su funcionamiento.

Además de este personal operativo, el COVAM cuenta con un grupo especializado encargado de la planificación y preparación de las actividades en las que participarán las unidades destacadas, así como del CEM. Asimismo, se dispone de personal dedicado al mantenimiento de las bases de datos y sistemas tecnológicos del centro, asegurando así la actualización y fiabilidad de la información utilizada en las operaciones de vigilancia y seguridad marítima.

Su trabajo se desarrolla en una amplia sala, representada en la Figura 2-13, en la que están los sistemas informáticos a través de los cuales el COVAM recibe información principalmente. Uno de los elementos empleados para el seguimiento del tráfico marítimo, aunque no el único, es el sistema automático de identificación (AIS, *Automatic Identification System*), que están obligados a montar la mayor parte de los buques en la mar [36].



Figura 2-13 Sala del COVAM desde donde se controlan las aguas españolas o de especial interés [36]

2.4.3 Datos AIS

El Sistema de Identificación Automática (AIS) es una tecnología de seguimiento y comunicación que utiliza transceptores en los buques y es empleada por los Servicios de Tráfico Marítimo (VTS) para la identificación y localización de embarcaciones. Su propósito principal es mejorar la seguridad en la navegación al facilitar la identificación de los buques y ayudar en la prevención de colisiones. Además, el AIS simplifica el intercambio de información y proporciona datos adicionales que contribuyen a evitar abordajes.

El sistema opera en las bandas de frecuencia VHF marítimas y transmite información como la identidad del buque, posición, rumbo y velocidad. Esta información es recibida por otras embarcaciones equipadas con AIS y por estaciones costeras, permitiendo una representación en tiempo real del tráfico

marítimo en la zona. El AIS complementa al radar, ofreciendo capacidades similares y, en algunos casos, mejorando la imagen del radar existente.

El AIS transmite varios tipos de información [39]:

- Datos estáticos: Incluyen el nombre del buque, número MMSI, distintivo de llamada, número OMI, dimensiones y tipo de barco. Esta información se introduce durante la instalación del dispositivo y solo necesita actualizarse si se modifican las características principales del buque.
- Datos dinámicos: Comprenden la posición, rumbo, velocidad y estado de navegación del buque. Estos datos se actualizan constantemente mediante sensores a bordo.
- Información del viaje: En el caso de los sistemas AIS de Clase A, se incluye información como el destino, rumbo y hora estimada de llegada (ETA).

Existen dos clases principales de transpondedores AIS [40]:

1. Clase A: Obligatoria para buques de arqueo bruto superior a 300 toneladas en viajes internacionales y para todos los buques de pasaje, independientemente de su tamaño. Estos transpondedores transmiten información con más frecuencia en comparación con los de Clase B.
2. Clase B: Diseñada para embarcaciones no sujetas al Convenio SOLAS, como embarcaciones de recreo y yates. Ofrecen los beneficios de seguridad y navegación del AIS con una instalación más sencilla y de menor coste que los dispositivos de Clase A.

El AIS ofrece múltiples beneficios en la navegación, destacando su capacidad para prevenir colisiones, ya que proporciona información detallada sobre la posición y movimiento de otras embarcaciones, permitiendo a los navegantes tomar decisiones informadas para evitar incidentes. Además, facilita la gestión del tráfico marítimo, permitiendo a las autoridades portuarias y los servicios de tráfico monitorizar y regular de manera más eficiente el flujo de embarcaciones en áreas congestionadas. En situaciones de emergencia, el AIS es una herramienta fundamental en las operaciones de búsqueda y rescate, ya que facilita la localización de embarcaciones en peligro, optimizando los tiempos de respuesta. Asimismo, desempeña un papel clave en la monitorización ambiental, permitiendo a las autoridades rastrear actividades pesqueras y detectar posibles infracciones o prácticas ilegales, contribuyendo así a la sostenibilidad y seguridad en el ámbito marítimo. Es importante destacar que, aunque el AIS es una herramienta valiosa para la seguridad marítima, su uso indebido o la manipulación de los datos transmitidos pueden comprometer la seguridad en el mar. Por ejemplo, se han reportado casos de embarcaciones que manipulan su información AIS para ocultar su verdadera posición o actividad, lo que representa un riesgo para la seguridad marítima y puede utilizarse para eludir sanciones o realizar actividades ilegales [41].

En resumen, el Sistema de Identificación Automática es una herramienta esencial en la navegación moderna, proporcionando información crítica para la seguridad en la mar. Su correcta implementación y uso son fundamentales para garantizar operaciones marítimas seguras y responsables.

2.5 Trabajos previos

El presente Trabajo de Fin de Grado se enmarca en una línea de investigación desarrollada en el Centro Universitario de la Defensa en la Escuela Naval Militar (CUD-ENM), la cual tiene como objetivo la aplicación de técnicas de inteligencia artificial y análisis de trayectorias en el ámbito marítimo. Esta línea ha sido explorada en trabajos previos que han sentado las bases para la mejora de los sistemas de predicción del tipo de buque a partir de datos cinemáticos y de trayectoria.

Se observó que la calidad de los datos AIS usados para entrenar a los modelos se veía afectada por el hecho de que los datos AIS tienen ciertas secciones que se rellenan manualmente por el personal de los buques. Estos datos no siempre se rellenan en concordancia con la realidad, siendo el campo *shiptype* uno de los más afectados por esto y complicando cualquier entrenamiento de IA debido a la incompletitud o poca fiabilidad de los datos aportados.

En línea de lo anterior, se planteó una línea TFG cuyo objetivo era tratar de mejorar la calidad de estos datos tratando de predecir con suficiente precisión el contenido del campo *shiptype*. A lo largo de los últimos años, se han desarrollado cuatro TFG en el CUD-ENM que han contribuido al progreso de esta investigación, abordando desde la recopilación y tratamiento de datos AIS hasta la implementación de modelos de ML para la clasificación de embarcaciones. En estos estudios, se han explorado diferentes metodologías para optimizar la precisión y eficiencia computacional de los sistemas de predicción, estableciendo un marco sobre el cual se sustenta el presente trabajo.

En [12], se empleó *Random Forest* con datos históricos preprocesados obtenidos del AIS, explorando diferentes configuraciones para mejorar la precisión en la clasificación del *shiptype*. Se realizaron variaciones en el algoritmo para la obtención de la máxima calidad de la predicción. En [42] se continúa el trabajo anterior, enfocándose en mejorar el preprocesado de datos y optimizar los tiempos de ejecución del modelo de predicción del tipo de buque. Se revisó la selección de parámetros utilizada previamente y se realizaron experimentos más exhaustivos para validar su robustez en distintos escenarios. Como resultado, se logró aumentar la confianza y fiabilidad de las predicciones, reforzando la idoneidad del enfoque inicial. Los datos estáticos adquirieron gran relevancia para la predicción en ambos trabajos.

En [3], se amplían las investigaciones previas al analizar el impacto del área de actividad del buque en la calidad de la predicción del tipo de barco. Se estudió cómo integrar esta información en los modelos de aprendizaje supervisado empleados y se realizaron múltiples experimentos para evaluar su efectividad. Los resultados confirman que el uso de datos geoespaciales mejora la precisión de la predicción, especialmente para ciertos tipos de buques.

Finalmente, el estudio realizado en [1] supone una evolución clave en la línea de investigación, ya que pasa de utilizar únicamente datos AIS estáticos, dinámicos y de área de actividad a incorporar información de trayectorias con hasta 67 parámetros, lo que enriquece significativamente el conjunto de características utilizadas en la predicción. Esto consume muchísimos recursos, dando lugar al desafío actual: mejorar la eficiencia computacional para procesar grandes volúmenes de datos de manera rápida y efectiva.

3 DESARROLLO DEL TFG

El presente capítulo se centra en la optimización del sistema de predicción, detallando los aspectos técnicos y metodológicos empleados. Se inicia con la descripción del entorno de trabajo y las herramientas utilizadas, justificando la elección de *Python*, *Jupyter Lab* y *cuDF* para el procesamiento optimizado de datos en GPU. Posteriormente, se explicará la construcción del modelo de predicción, empleando los algoritmos *Random Forest* y *XGBoost*, con un enfoque en la mejora de la eficiencia computacional.

Además, se abordará el tratamiento y preprocesamiento de los datos AIS, un aspecto crucial dado que el modelo se nutre de datos en tiempo real. Finalmente, se presentará la implementación del preprocesamiento de los datos y entrenamiento del modelo, analizando el impacto de la optimización en GPU en comparación con la ejecución tradicional en CPU.

3.1 Entorno de trabajo y herramientas empleadas

El desarrollo de este sistema de predicción de tipo de buque requiere una infraestructura computacional que permita gestionar grandes volúmenes de datos de manera eficiente. En este apartado se describen las principales herramientas y tecnologías utilizadas para la implementación del modelo, desde el hardware empleado hasta los entornos de desarrollo y las bibliotecas utilizadas para la optimización del procesamiento de datos.

3.1.1 Hardware

Se ha empleado un equipo de alto rendimiento que permite optimizar el procesamiento de grandes volúmenes de datos. Dado que el objetivo principal del trabajo es mejorar la eficiencia computacional del sistema de predicción de tipo de buque, se ha optado por un hardware que favorezca el uso de procesamiento paralelo en GPU y una alta capacidad de almacenamiento y memoria para la manipulación de datos.

El equipo utilizado cuenta con las siguientes especificaciones técnicas:

- Procesador: Intel Core i9-14900K (24 núcleos)
- Memoria RAM: 192GB DDR5, optimizada para la ejecución de cargas de trabajo intensivas y procesamiento de grandes *datasets*.
- Tarjeta gráfica (GPU): NVIDIA RTX 4090 Ventus (con 24GB de memoria VRAM).
- Almacenamiento: Discos SSD NVMe, permitiendo una mayor velocidad de lectura y escritura de datos en comparación con los discos tradicionales, optimizando el acceso a los conjuntos de datos de entrenamiento.

La elección de este hardware responde a la necesidad de procesar grandes volúmenes de datos de manera eficiente, reduciendo significativamente los tiempos de ejecución en comparación con arquitecturas basadas únicamente en CPU. La implementación de bibliotecas de cómputo acelerado en GPU, como *cuDF* [43] y *RAPIDS* [44], permite aprovechar la capacidad de paralelización de la tarjeta gráfica (RTX 4090), lo que resulta fundamental para mejorar el rendimiento en la manipulación y análisis de datos AIS.

En comparación con un procesamiento convencional en CPU, el uso de GPU permite una reducción drástica en los tiempos de cómputo, especialmente en el procesamiento de datos y en el entrenamiento de modelos basados en *Random Forest* y *XGBoost* [45]. Esta ventaja es especialmente relevante en entornos donde se requiere una rápida actualización y análisis de datos en tiempo real.

Por lo tanto, la combinación de un procesador de alto rendimiento, una gran capacidad de memoria RAM y una GPU optimizada para cómputo paralelo resulta clave para alcanzar los objetivos del proyecto, asegurando una ejecución eficiente del modelo y facilitando la futura integración de técnicas avanzadas de procesamiento en tiempo real.

3.1.2 Jupyter Lab

Para el desarrollo y experimentación con los modelos de predicción, se ha usado *Jupyter Lab* [46]. Es un sistema multiplataforma y multilenguaje que permite la codificación organizada por celdas y la visualización de datos vía web.

Presenta ciertas ventajas frente a otros entornos que han hecho que sea la mejor opción para el desarrollo de este TFG. Algunas de estas ventajas son la interactividad y flexibilidad que *Jupyter Lab* ofrece gracias a la opción de ejecutar el código en celdas, facilitando la depuración, prueba de modelos y soporte a numerosos lenguajes de programación, además de permitir que se ejecuten *Notebooks* en servidores remotos.

3.1.3 Python

El lenguaje utilizado es *Python* [47], debido a su amplia adopción en el campo de la inteligencia artificial y el aprendizaje automático. *Python* es un lenguaje de programación ampliamente usado en el desarrollo de software, en trabajos con datos y en ML. Está desarrollado con código de fuente abierta [47]. Se han empleado diversas bibliotecas para cada fase del desarrollo del modelo:

- *NumPy* [48] y *pandas* [49]: Manipulación y análisis de datos.
- *cuDF* [43]: Alternativa acelerada en GPU para *pandas*.
- *Random Forest* [50]: Algoritmo heredado de TFG previos.
- *XGBoost* [51]: Algoritmo basado en árboles optimizado para alto rendimiento.

3.1.4 RAPIDS

RAPIDS [52] es un ecosistema de bibliotecas *open source* desarrollado por NVIDIA que permite acelerar de manera significativa los flujos de trabajo en ciencia de datos y ML, aprovechando la potencia del procesamiento paralelo de las GPU. La integración de *RAPIDS* mejora teóricamente la eficiencia computacional mediante la aceleración de la GPU consiguiendo velocidades entre 5 y 50 veces mayores a la vez que es compatible con las bibliotecas más populares.

RAPIDS está construido sobre la arquitectura *CUDA* de NVIDIA, lo que le permite aprovechar el paralelismo masivo de las GPU. Esto significa que muchas operaciones, que en un entorno tradicional con CPU se realizan de forma secuencial, pueden ejecutarse en paralelo, reduciendo drásticamente los tiempos de procesamiento. Esta aceleración es especialmente relevante cuando se trabaja con grandes volúmenes de datos o algoritmos computacionalmente intensivos.

En la Figura 3-1 se representa una pila integral que está compuesta por varios niveles que se relacionan entre sí, permitiendo la ejecución eficiente de aplicaciones de ciencia de datos e inteligencia artificial. En el nivel superior están las aplicaciones de ciencia de datos, que se benefician directamente de estas optimizaciones. Justamente a continuación, el segundo nivel representa las interfaces para la integración directa con aplicaciones ya existentes, como *Apache Spark* y bibliotecas populares como *XGBoost*. Más abajo, en el tercer nivel, se encuentran bibliotecas especializadas del ecosistema RAPIDS, como *cuDF* para el manejo eficiente de *DataFrame* (DF), *cuML* para aprendizaje automático y *cuGraph* para el procesamiento de grafos.

El cuarto nivel resalta la capa de software CUDA, fundamental para aprovechar la aceleración con GPU. Esta capa permite que las bibliotecas superiores utilicen las capacidades del hardware para ejecutar operaciones de manera más rápida y eficiente en comparación con una CPU convencional. Finalmente, en la base de la imagen se observa el hardware compatible con esta tecnología, mostrando los diferentes entornos posibles de ejecución: *Cloud*, *Data Center* y *Workstation*.

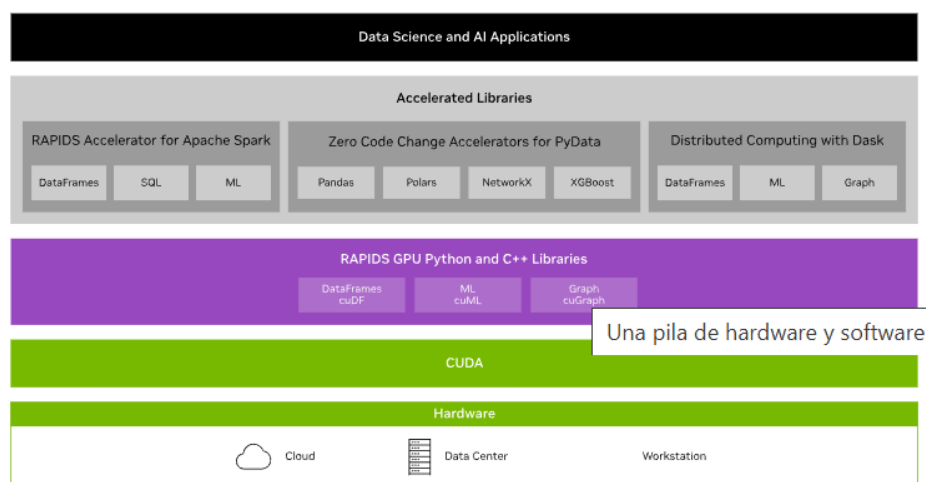


Figura 3-1 Pila de *hardware* y *software* optimizada para el proceso de ciencia de datos con RAPIDS [52]

3.1.5 *cuDF*

cuDF [43] es una biblioteca diseñada para trabajar con datos tabulares DF del mismo modo que *pandas*, pero con la capacidad de aprovechar el procesamiento masivo paralelo de las GPU. Esto se logra gracias a la arquitectura de cómputo de las GPU, que permite ejecutar miles de operaciones simultáneamente en lugar de seguir un enfoque secuencial como ocurre en las CPU.

La razón clave por la que *cuDF* mejora la eficiencia computacional es la paralelización masiva. Las operaciones sobre un DF de *pandas* se ejecutan secuencialmente o, en el mejor de los casos, con cierto grado de paralelismo mediante múltiples hilos. En una GPU con *cuDF*, las operaciones sobre los datos se dividen en múltiples fragmentos, que se ejecutan simultáneamente en miles de núcleos de procesamiento, permitiendo una reducción drástica en los tiempos de ejecución. Además, las GPU están diseñadas para trabajar con grandes volúmenes de datos y cuentan con una arquitectura de memoria de alta velocidad en comparación con las CPU. *CuDF* aprovecha esta ventaja optimizando el acceso a la memoria y minimizando los cuellos de botella asociados a la transferencia de datos entre memoria RAM y la unidad de procesamiento.

3.2 Datos estáticos

Los TFG anteriores se enfocaron en el procesado de los datos estáticos, siendo objeto de estudio en [12] mientras que en [1] se depuraron estos procesos. En la Figura 3-2 se muestra el segmento de código de [1] que tarda más tiempo en ejecutarse de todo el código que se emplea para la preparación de datos estáticos (*prep_statics*). El código de este fragmento lee los archivos y los almacena en un DF. Teniendo

en cuenta que es un fragmento que solo tarda 1 minuto 1 segundo (*Wall time* en la Figura 3-2) en ejecutarse, es un tiempo más que asumible. Dado que el objetivo de este TFG es mejorar la eficiencia computacional, carece de sentido centrar esfuerzos en esta parte.

```
%%time
# Explicación de como se genera raw_static_data.csv
# Con pocos datos es viable trabajar con todos a la vez (útil para EDA)
# Almacena los nombres solo los archivos que contienen "static" en el nombre
lista_archivos_static = [archivo for archivo in os.listdir('csv') if "static" in archivo]
# Para ver los nombres de los archivos: print(lista_archivos_static)
# Crea un DataFrame vacío para ir almacenando los datos de todos los csv estáticos
df_raw_static = pd.DataFrame()
# Itera sobre los archivos "static" y los carga en el DataFrame
for archivo_static in lista_archivos_static:
    df_temporary = pd.read_csv(os.path.join('csv', archivo_static))
    df_raw_static = pd.concat([df_raw_static, df_temporary], ignore_index=True)
# Ahora df_raw_static contiene todos los datos de los archivos estáticos

CPU times: user 46.8 s, sys: 15 s, total: 1min 1s
Wall time: 1min 1s
```

Figura 3-2 Almacenamiento de los datos de estáticos en *df_raw_static*

Visto que los tiempos de ejecución en lo que respecta al procesamiento de los datos estáticos ya son lo suficientemente reducidos, en este TFG nos limitaremos a probar otros algoritmos de ML llevando a cabo una repetición en el análisis y preparación de los datos. Para ello se utilizará *XGBoost* (más adelante se explicará el proceso). Con ello se pretende mejorar la precisión en la predicción del tipo de buque a partir de los datos estáticos cambiando el algoritmo de aprendizaje usado. El algoritmo *XGBoost* supera al *Random Forest* en muchos aspectos, el más reseñable es que *XGBoost* da siempre más importancia a reducir el coste de un modelo mientras que *Random Forest* aumenta de tamaño sin tener esto en cuenta. Más allá, *XGBoost* permite un aprendizaje secuencial con árboles que corrigen los errores de los anteriores en vez de ser independientes como en *Random Forest*. Por último, *XGBoost* está diseñado para la velocidad, siendo perfecto para grandes conjuntos de datos. Aunque para los datos estáticos no sea muy importante este aspecto (los tiempos de ejecución en este caso son muy reducidos), para el empleo de datos dinámicos se prevé una mayor diferencia [53].

3.2.1 Definición de los campos de los datos estáticos y parámetros extraídos

Para la predicción del tipo de barco se trabajará sobre los datos AIS proporcionados al CUD-ENM por el COVAM que ya han sido procesados. Para el análisis de los datos y su procesamiento es imprescindible conocer qué datos nos proporciona el AIS y qué representan. Los datos estáticos muestran características del buque. Estos datos se introducen manualmente por el armador o patrón de dicho buque y se pueden cambiar en cualquier momento [54]. En la Tabla 3-1 se muestran los datos estáticos ya tratados que incluye el DF que se usa en el desarrollo del código.

Campos	Descripción
<i>Timestamp</i>	Hora de recepción del mensaje AIS
<i>MMSI</i>	<i>Maritime Mobile Service Identity</i>
<i>IMO</i>	Número de Identificación de la OMI
<i>To_bow</i>	Distancia de la antena a proa (m)
<i>To_stern</i>	Distancia de la antena a popa (m)
<i>To_port</i>	Distancia de la antena al costado de babor (m)
<i>To_starboard</i>	Distancia de la antena al costado de estribor (m)
<i>Draught</i>	Calado (m)
<i>Shiptype</i>	Tipo de buque

Tabla 3-1 Campos del fichero de datos estáticos (csv)

Para mayor comprensión de los parámetros estáticos detrás de la predicción, en la Tabla 3-2 se describe el significado de todos los parámetros estáticos usados para entrenar el modelo de predicción usado. Como veremos a lo largo de los resultados, los parámetros estáticos son los más relevantes a la hora de la predicción, por tanto, es de vital importancia que se entiendan a la perfección. Estos parámetros han sido extraídos y calculados a partir de los campos definidos en la Tabla 3-1 de los cuales, los campos *To_bow*, *To_stern*, *To_port*, *To_starboard* y *Draught* también serán usados para el entrenamiento del modelo y predicción del *shiptype*.

Parámetro estático	Descripción
<i>len</i>	Eslora
<i>wid</i>	Manga
<i>ldwid</i>	Cociente entre eslora y manga
<i>ldivd</i>	Cociente entre eslora y calado
<i>wdivd</i>	Cociente entre manga y calado
<i>area</i>	Área aproximada de la cubierta (eslora x manga)
<i>grith</i>	Suma de la eslora y manga
<i>aml</i>	Área longitudinal sumergida (eslora x calado)
<i>amt</i>	Área transversal sumergida (manga x calado)
<i>Vs</i>	Volumen sumergido (manga x eslora x calado)
<i>Aol</i>	Cociente entre la distancia desde la antena a proa y la eslora total

Tabla 3-2 Descripción de los parámetros estáticos

3.2.2 Métricas empleadas

La biblioteca *XGBoost* ofrece la función *classification_report* mediante la cual se generan las métricas usadas posteriormente para la evaluación de los resultados. Estas métricas serán empleadas para analizar los resultados obtenidos en cada experimento y, por tanto, aparecerán en todas las figuras que traten sobre los resultados. A continuación, se describen y ejemplifican cada una de ellas para su mejor comprensión:

- **Exactitud (*Accuracy*):** Representa el porcentaje de predicciones correctas realizadas por el modelo. Se obtiene dividiendo el número de aciertos entre el total de predicciones. Aunque es una métrica sencilla y ampliamente utilizada, su interpretación debe ser cuidadosa en conjuntos de datos desbalanceados. Por ejemplo, si en un conjunto de datos hay 5000 buques, de los cuales 4500 son de carga y solo 500 son de otro tipo, un modelo que clasifique todos los buques como de carga alcanzaría una exactitud del 90%, a pesar de no identificar correctamente a los petroleros, remolcadores o pesqueros. Esto indica que, aunque la exactitud sea alta, el modelo no es eficaz en la clasificación de las clases minoritarias.
- **Precisión (*Precision*):** Mide la proporción de predicciones correctas dentro de todas las predicciones realizadas para una clase específica. Se calcula como la cantidad de elementos correctamente clasificados en una clase dividida por el total de elementos que el modelo ha identificado como pertenecientes a esa clase. Por ejemplo, si el modelo clasifica 200 barcos como petroleros, pero solo 120 de ellos lo son realmente, la precisión para esa clase será del 60%. En cambio, si clasifica 80 barcos como petroleros y todos ellos son correctos, la precisión sería del 100%, aunque haya otros petroleros que el modelo no haya identificado como tales. Para equilibrar este aspecto, se utiliza la métrica que se explica a continuación.
- **Exhaustividad (*Recall*):** Evalúa la capacidad del modelo para identificar correctamente todos los elementos de una clase. Se calcula dividiendo el número de elementos correctamente clasificados en una clase por el total de elementos que realmente pertenecen a esa clase. Por ejemplo, si en el conjunto de datos hay 150 barcos pesqueros y el modelo solo clasifica correctamente 90 de ellos, la exhaustividad para esta clase será del 60%. Esto significa que el 40% de los pesqueros han sido clasificados erróneamente como otro tipo de buque.
- **Valor F1 (*F1-Score*):** Es una métrica que combina precisión y exhaustividad en un único valor, proporcionando una visión más equilibrada del rendimiento del modelo. Se calcula como la media armónica entre ambas métricas, lo que permite evitar interpretaciones erróneas que podrían surgir si se analiza solo una de ellas. El *F1-score* es especialmente útil cuando el conjunto de datos está desbalanceado o cuando se necesita un buen equilibrio entre la identificación correcta de clases y la minimización de falsos positivos. Por ejemplo, si un modelo tiene una precisión del 80% pero una exhaustividad del 50% en la clasificación de buques de pasajeros, el *F1-score* reflejará que, aunque el modelo es bastante preciso cuando hace una predicción, todavía deja sin clasificar correctamente a una gran cantidad de buques de esta categoría.

La Figura 3-3 refleja el código usado para la generación de las métricas y la evaluación de la calidad en la predicción del modelo con el que se obtendrán los resultados para los experimentos estáticos realizados.

```
# Evaluar el modelo
accuracy_val = accuracy_score(y_test_np, pred_np)
print(f"Precisión del modelo: {accuracy_val:.4f}")

# Reporte de métricas
reporte = classification_report(y_test_np, pred_np, digits=2)
print("\nReporte de Clasificación:\n", reporte)
```

Figura 3-3 Obtención de precisión y métricas

3.2.3 Experimentos estáticos

Al haber cambiado la máquina sobre la que se ejecutarán los algoritmos, siendo más potente la actual, se debe establecer el punto de partida para poder llevar a cabo una comparación justa sin que el nuevo equipo influya en la previsible mejora de los tiempos de ejecución, ya que la precisión de los resultados no se espera que varíen ya que no dependen de la potencia del equipo usado. El punto de partida se establecerá ejecutando el código en [1] en la nueva máquina y registrando los resultados obtenidos, tanto de precisión como de tiempos de ejecución. Veíamos en el TFG predecesor que se usaba el algoritmo de aprendizaje automático supervisado *Random Forest*. En la Figura 3-4 observamos los resultados que se obtuvieron el año pasado en [1].

	precision	recall	f1-score	support
ShipType.Cargo	0.88	0.92	0.90	5072
ShipType.Fishing	0.81	0.81	0.81	1225
ShipType.Passenger	0.75	0.74	0.74	627
ShipType.Tanker	0.86	0.75	0.80	2029
ShipType.Tug	0.83	0.85	0.84	1262
accuracy			0.86	10215
macro avg	0.83	0.82	0.82	10215
weighted avg	0.86	0.86	0.85	10215

Figura 3-4 Resultados obtenidos con *Random Forest* de [1]

Comparando los resultados del año pasado con los resultados obtenidos este año, donde se han utilizado más buques para el entrenamiento (representados en la Figura 3-5), se observa que hay una ligera diferencia. Esta ínfima diferencia puede deberse a que el número de la columna *support* (tamaño del conjunto de pruebas) y el tamaño de los datos de entrenamiento son mayores ahora que anteriormente. Al disponer de más buques, es factible que la predicción mejore.

	precision	recall	f1-score	support
ShipType.Cargo	0.89	0.93	0.91	5508
ShipType.Fishing	0.84	0.87	0.85	1410
ShipType.Passenger	0.79	0.76	0.78	733
ShipType.Tanker	0.89	0.77	0.82	2207
ShipType.Tug	0.86	0.87	0.87	1484
accuracy			0.87	11342
macro avg	0.85	0.84	0.85	11342
weighted avg	0.87	0.87	0.87	11342

Figura 3-5 Resultados del Experimento estáticos 0

Una vez establecido el punto de partida, que será la precisión de 0.87 observada en la Figura 3-5, se tratará de mejorar este valor utilizando el algoritmo *XGBoost*. Este ofrece a los usuarios la capacidad de ajustar los parámetros en correspondencia con los datos. Esto nos orienta hacia el siguiente objetivo del desarrollo, ajustar el algoritmo para conseguir la mayor precisión posible sin llegar al sobreajuste del que se hablaba en el apartado 2.2.

```

# Asegurar que shiptype sea numérico
encoder = LabelEncoder()
df_final_static["shiptype"] = encoder.fit_transform(df_final_static["shiptype"])

# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_final_static, test_size=0.2, random_state=43, stratify=df_final_static["shiptype"])

# Separar características y variable objetivo
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train["shiptype"]
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test["shiptype"]

# Creación del modelo con los datos correctos
x_train_dmatrix = xgb.DMatrix(x_train, label=y_train)
x_test_dmatrix = xgb.DMatrix(x_test, label=y_test)

# Definir parámetros
parametros = {
    "booster": "gbtree",
    "num_class": len(y_train.unique()),
    "max_depth": 5,
    "eta": 0.8,
    "objective": "multi:softmax",
    "nthread": 2
}

rondas = 100
modelo = xgb.train(parametros, x_train_dmatrix, rondas) # Entrenar el modelo
prediccion = modelo.predict(x_test_dmatrix) # Hacer predicciones

```

Figura 3-6 Configuración del algoritmo XGBoost

En la Figura 3-6 se puede observar el código que utiliza *XGBoost* y los parámetros utilizados para realizar la predicción. Estos parámetros describen [55]:

- *Booster*: Tipo de modelo de clasificación, por defecto será *gbtree*, que construye árboles de decisión en serie corrigiendo cada uno los errores del anterior. A mayores se prueba el *dart* (*Dropout Additive Regression Trees*) que descarta algunos árboles aleatoriamente para evitar que el modelo dependa demasiado de ciertos patrones.
- *num_class*: solo se utiliza para indicar cuántas clases distintas hay en el conjunto de datos, debe coincidir con el número de clases en las que se quiere clasificar según los parámetros estudiados.
- *max_depth*: Profundidad o cantidad de nodos de bifurcación de los árboles de decisión usados en el entrenamiento. Presenta el problema del sobreajuste. Rango: $[0, \infty]$.
- *eta*: Tasa de aprendizaje del modelo que define la rapidez con la que el algoritmo aprende de cada iteración. Un valor más bajo de *eta* provoca un aprendizaje más lento reduciendo la contribución de cada árbol del conjunto. Rango: $[0, 1]$.
- *objective*: Indica qué tipo de función va a usar dependiendo del tipo de tarea. En este caso se utiliza: *multi:softmax* usado para clasificación multiclase como es el caso.
- *nthread*: Número de hilos computacionales. Se refiere a los núcleos del procesador del equipo.
- *rondas*: Número de iteraciones que se realizarán. A mayor número de rondas, mejores resultados, pero requiere de mayores tiempos de entrenamiento y conlleva un riesgo de sobreajuste. Rango: $[0, \infty]$.
- *test_size*: en la línea seis del código, se indica la proporción de datos que se usarán para la prueba del modelo con el que se evalúa la precisión con la que predice los *shiptype*. Rango: $[0, 1]$.

Con la implementación de *XGBoost*, se ha usado un mapeo que se mantendrá de aquí en adelante para los distintos *shiptype*. Este mapeo según la codificación de la Tabla 3-3 se ha realizado para facilitar el tratamiento de los datos de esta columna y mejorar la eficiencia, decidiéndose cambiar el tipo de datos *object* a un número entero. Se modificará en el archivo CSV *final_static_data_jan* y, desde el comienzo, se usarán estos códigos para representar los tipos de buque.

Mapeo de *shiptype* a numérico

ShipType.Cargo → 0

ShipType.Fishing → 1

ShipType.Passenger → 2

ShipType.Tanker → 3

ShipType.Tug → 4

Tabla 3-3 Mapeo de *shiptype* de *string* a código numérico

Los parámetros mostrados en la Figura 3-6 son los parámetros iniciales. A partir de ellos se ha ajustado el modelo de manera empírica, con una metodología de prueba y error basándonos siempre en unos valores lógicos y que eviten el sobreajuste. En el *Experimento estáticos 0*, denominado así el punto de partida, se obtienen los siguientes resultados que se muestran en la Figura 3-7, donde puede verse que se consigue una precisión de un 88%, valor de precisión que ya supera por una décima a la obtenida con *Random Forest*.

Precisión del modelo: 0.8813

Reporte de Clasificación:				
	precision	recall	f1-score	support
0	0.90	0.93	0.91	4003
1	0.88	0.88	0.88	988
2	0.81	0.80	0.80	527
3	0.87	0.79	0.82	1597
4	0.88	0.87	0.87	1057
accuracy			0.88	8172
macro avg	0.87	0.85	0.86	8172
weighted avg	0.88	0.88	0.88	8172

Figura 3-7 Resultados del Experimento estáticos 1

El primer parámetro que cambiaremos será el *max_depth*, lo estableceremos en 10 para el *Experimento estáticos 2*. Como se observa en la Figura 3-8, la calidad de la predicción media (*Accuracy*) mejora ínfimamente, no obstante, hay que tener en cuenta que se redondea hacia arriba a partir de la mitad, por lo que no es completamente exacto. Al seguir subiendo el *max_depth* hasta 11 o 12 no mejora, sino que empeora por lo que se mantiene *max_depth=10* como la mejor opción.

Precisión del modelo: 0.8836

Reporte de Clasificación:				
	precision	recall	f1-score	support
0	0.90	0.94	0.92	4003
1	0.87	0.88	0.87	988
2	0.81	0.79	0.80	527
3	0.88	0.80	0.84	1597
4	0.87	0.86	0.87	1057
accuracy			0.88	8172
macro avg	0.87	0.85	0.86	8172
weighted avg	0.88	0.88	0.88	8172

Figura 3-8 Resultados del Experimento estáticos 2

El *Experimento estáticos 3* consiste en optimizar el parámetro *eta*. Primero, se establece a 0.3 con lo que mejora notablemente hasta 0.8857, se sigue probando con distintos valores, incluso con 0.1 pero, finalmente, es en 0.2 donde se encuentra el máximo con una precisión del modelo de 0.8857 como se observa en la Figura 3-9.

Precisión del modelo: 0.8857

Reporte de Clasificación:				
	precision	recall	f1-score	support
0	0.90	0.94	0.92	4003
1	0.87	0.89	0.88	988
2	0.80	0.79	0.79	527
3	0.89	0.79	0.84	1597
4	0.88	0.87	0.87	1057
accuracy			0.89	8172
macro avg	0.87	0.86	0.86	8172
weighted avg	0.89	0.89	0.88	8172

Figura 3-9 Resultados del Experimento estáticos 3

El siguiente experimento consiste en modificar el número de rondas realizadas. Se elevará el número primeramente hasta 500 rondas con lo que se mejora la predicción hasta un 0.8896, rozando el 89%. No obstante, hay que tener en cuenta que los tiempos de ejecución aumentan a medida que se aumenta el número de rondas. Finalmente, se encuentra el máximo en 650 rondas y habiendo conseguido sobrepasar el 89% de aciertos en la predicción. En la Figura 3-10 se observan los resultados detallados.

Precisión del modelo: 0.8905

Reporte de Clasificación:				
	precision	recall	f1-score	support
0	0.91	0.94	0.92	4003
1	0.87	0.89	0.88	988
2	0.81	0.79	0.80	527
3	0.89	0.81	0.85	1597
4	0.88	0.87	0.88	1057
accuracy			0.89	8172
macro avg	0.87	0.86	0.87	8172
weighted avg	0.89	0.89	0.89	8172

Figura 3-10 Resultados del Experimento estáticos 4

Por último, en el *Experimento estáticos 5*, se modifica el porcentaje de datos usados para el entrenamiento y para la prueba del modelo. Tras probar distintas combinaciones como el 15%, 25% y 30% de datos reservados para la prueba, se observa que la mejor opción es la inicial, con un 20% de datos para la prueba, por lo que se mantiene este parámetro inicial.

Tras las mejoras fallidas durante el *Experimento estáticos 5*, la precisión en la predicción del modelo final de *XGBoost* será del 89.05% (más detallada en la Figura 3-10) habiendo superado en un 3% a la calidad de la predicción del algoritmo *Random Forest* usado en [1]. Llegados a este punto, se cerraría el estudio de las posibles mejoras en el campo de datos estáticos, habiendo comprobado que, como era previsible, el margen de mejora era escaso.

3.3 Datos dinámicos

En esta parte del TFG, el esfuerzo se ha centrado en la optimización del procesamiento de datos dinámicos, la parte del código que inicialmente consumía la mayor cantidad de recursos y tiempo debido a la elevada cantidad de información y operaciones de procesamiento requeridas. Por ello, se han dirigido los esfuerzos a modificar el código existente y a incorporar cálculos de parámetros de forma directa, con el objetivo de adaptar y optimizar los procesos para que sean compatibles con la ejecución en GPU. Este

enfoque permite paralelizar las operaciones, reduciendo significativamente los tiempos de ejecución y aumentando la eficiencia computacional.

Este objetivo debe cumplirse sin comprometer la calidad de la predicción. Inicialmente se hará la predicción con *Random Forest*. Una vez se haya confirmado que la calidad de la predicción se mantiene, pasaremos a tratar de mejorarla mediante el modelo de *XGBoost* con los parámetros con los que se optimizó el algoritmo durante los experimentos con datos estáticos para el entrenamiento y predicción del *shiptype*.

El análisis de los datos dinámicos presenta mayores desafíos debido a la forma en que han sido proporcionados. En este caso, la información de cada buque se encuentra almacenada en archivos independientes denominados en función de su MMSI, donde se recopila toda la información dinámica del buque correspondiente a un mes determinado.

3.3.1 Definición de los campos de los datos dinámicos y parámetros extraídos

Los datos dinámicos usados se proporcionan agrupados en una carpeta llamada *Dynamic_by_MMSI_month*. Se accederá a los distintos archivos de manera iterativa ya que esto permitirá la escalabilidad a mayor tamaño de datos. No obstante, se realizará un último experimento accediendo a los datos de manera unificada mediante un DF que contenga todos los datos. Además, como se detallará más adelante, los campos disponibles no ofrecen de manera intuitiva información relevante, por lo que requieren un preprocesamiento previo antes de su análisis. Estos archivos contienen para todos ellos los mismos campos, los cuales se describen en la Tabla 3-4.

Campos	Descripción
<i>Timestamp</i>	Hora de recepción del mensaje AIS
<i>MMSI</i>	<i>Maritime Mobile Service Identity</i>
<i>Lat</i>	Latitud GPS en el momento de transmisión
<i>Lon</i>	Longitud GPS en el momento de transmisión
<i>Speed</i>	Velocidad instantánea
<i>Course</i>	Rumbo sobre fondo instantáneo
<i>Heading</i>	Dirección de la proa

Tabla 3-4 Campos de los datos dinámicos proporcionados en el archivo csv

3.3.2 Detección de cuellos de botella

Habiendo observado que los tiempos de ejecución eran extremadamente largos, se hizo un análisis de los cuellos de botella. En la Figura 3-11 se representa el tiempo de ejecución de dicho bucle de procesamiento de los datos.

Habiendo llegado a la conclusión de que el bucle principal consumía demasiado tiempo, se decidió probar con la paralelización de los procesos. Las GPU están diseñadas como motores de cálculo masivamente paralelo. A diferencia de las CPU, que optimizan la ejecución secuencial mediante sofisticadas técnicas de control y grandes memorias caché, las GPU incorporan cientos o incluso miles de núcleos simples [56].

Adicionalmente, dentro de este bucle, se concluyó en [1] que la generación y discretización de trayectorias por medio de las bibliotecas de *movingpandas* era lo que más tiempo de ejecución consumía.

Esta limitación se ha tratado de solucionar reemplazando el empleo de esta biblioteca e implementando directamente las funciones de interés.

3.3.3 Experimentos dinámicos

3.3.3.1 Experimento datos globales 0

Las pruebas iniciales se realizaron ejecutando el código desarrollado en [1] sin modificaciones, aprovechando que el servidor actual cuenta con mayores prestaciones. Este enfoque estableció un punto de partida homogéneo, permitiendo comparar de forma precisa las mejoras obtenidas tras la optimización, sin que las diferencias en capacidad de hardware influyeran en los resultados.

Los resultados mostrados en la Figura 3-11 representan el tiempo inicial invertido en la ejecución del código. El tiempo de ejecución es de 73 horas, 7 minutos y 22 segundos; evidenciando que el margen de mejora es grande. Este tiempo concuerda con los tiempos de ejecución en [1]. Se plantea que trasladar la carga de trabajo desde la CPU a la GPU es la estrategia óptima para mejorar el rendimiento en el procesamiento de datos dinámicos.

```
Processing MMSIs: 100%|██████████| 45367/45367 [73:07:22<00:00, 5.80s/it]
Existen 98 MMSI cuyo df contiene Nan.
Existen 2950 MMSI que no cumplen con la longitud requerida.
Existen 0 MMSI que no se han asociado a ningún shiptype.
```

Figura 3-11 Tiempos de ejecución del procesamiento de datos dinámicos en el Experimento datos globales 0

3.3.3.2 Aplicación directiva %load_ext cudf.pandas

Se ponen en marcha distintas iniciativas para transformar el código inicial en un código que trabaje con CUDA y GPU. Una de ellas es la planteada en el primer experimento, en el que se estudia la aplicación de una directiva que no precisaría de modificaciones en el código y que adapta de forma transparente al usuario la biblioteca *pandas* (la utilizada en el código de partida) a *cuDF*.

La directiva usada es una orden mágica de *Python* que carga una extensión proporcionada por la biblioteca *cuDF* y forma parte del ecosistema RAPIDS. Está diseñada para ofrecer operaciones de manipulación de datos aceleradas por GPU, imitando en gran medida la interfaz y funcionalidad de *pandas*.

Al ejecutar la directiva, se habilitan herramientas y funciones adicionales que facilitan la integración entre *cuDF* y *pandas*. Esto permite, por ejemplo, convertir DF de *pandas* a *cuDF* de forma transparente o aprovechar funcionalidades específicas que ayudan a migrar y adaptar flujos de trabajo basados en *pandas* hacia un entorno GPU. De esta manera, se facilita la transición de operaciones tradicionales en CPU a procesos paralelizados en GPU, mejorando el rendimiento en el procesamiento de grandes volúmenes de datos.

Esta extensión es especialmente útil en entornos de desarrollo interactivo, como los *notebooks* de *Jupyter*, ya que permite experimentar con código que puede beneficiarse de la aceleración por GPU sin abandonar la sintaxis familiar de *pandas* [43].

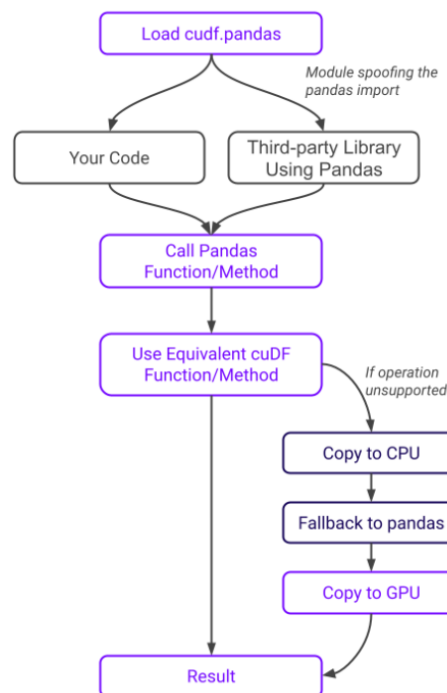


Figura 3-12 Funcionamiento de la directiva `%load_ext cudf.pandas` [57]

En la Figura 3-12 se representa el flujo de ejecución de la directiva `%load_ext cudf.pandas`. Las funciones y operaciones sobre tipos *proxy* se ejecutan en la GPU siempre que sea posible, y en la CPU en caso contrario, sincronizándose de forma interna según se requiera. Esto aplica a todas las operaciones con *pandas*, ya sea en el propio código del usuario o en bibliotecas de terceros.

Las funciones y operaciones sobre tipos *proxy* quiere decir que actúan como intermediarios. En lugar de ser implementaciones “reales” de *pandas*, estos *proxies* tienen la misma interfaz que *pandas* pero, internamente, redirigen la ejecución de las operaciones a la GPU o CPU según la posibilidad.

En *cudf.pandas*, cada objeto funciona como un intermediario que puede apuntar a un objeto en la GPU (*cuDF*) o en la CPU (*pandas*) en cualquier momento. Por defecto, se intenta ejecutar las operaciones en la GPU, copiando los datos desde la CPU si es necesario; y, si no es posible, se recurre a la CPU, trasladando la información desde la GPU cuando hace falta. Además, al usar *cudf.pandas* se activa de forma automática el modo de compatibilidad con *pandas* de *cuDF*, lo que garantiza el mantenimiento de convenciones propias de *pandas*, como el orden de clasificación predeterminado [57].

En el diagrama de la Figura 3-12 se representa el funcionamiento interno de esta directiva y, para una mayor comprensión, se pasa a explicar paso a paso:

1. **Carga del módulo *cudf.pandas*:**
 - Al ejecutar `%load_ext cudf.pandas`, lo que ocurre es que *cudf.pandas* se inyecta en el entorno y actúa como un reemplazo de *pandas*.
 - Este mecanismo se conoce como *module spoofing* (suplantación de módulos), ya que intercepta las importaciones de *pandas* en el código y redirige las llamadas a *cuDF*.
2. **Código del usuario o biblioteca de terceros usando *pandas*:**
 - El código y cualquier biblioteca de terceros que utilice *pandas* puede seguir funcionando sin cambios aparentes dado que cualquier llamada a *pandas* será automáticamente redirigida a *cuDF*, siempre que haya una función equivalente disponible.
3. **Llamada a función o método de *pandas*:**
 - Cuando el código (o biblioteca de terceros) intenta usar una función que resuelve *pandas*, el sistema primero intenta resolver la llamada usando la implementación de *cuDF*.

4. Uso de una función o método equivalente de *cuDF*:

- Si *cuDF* tiene una versión optimizada en la GPU para la función de *pandas*, la ejecución se lleva a cabo en la GPU sin la necesidad de intervención manual.
- Esto permite un aumento significativo en el rendimiento en comparación con la ejecución tradicional en CPU.

5. Manejo de operaciones no soportadas:

- No todas las funciones de *pandas* tienen una implementación directa en *cuDF*. Si una función específica no está soportada, los datos se copian a la CPU, se realiza una llamada a *pandas* en CPU y los resultados se copian de vuelta en la GPU.
- Este mecanismo permite que el código siga funcionando sin errores, aunque con una penalización en el rendimiento debido a la transferencia de datos entre la CPU y GPU.

6. Obtención del resultado final:

- Una vez que la operación se ha ejecutado correctamente, ya sea en *cuDF* (GPU) o en *pandas* (CPU), el resultado se devuelve al usuario.

El resultado del tiempo de ejecución al haber aplicado esta directiva no es el esperado, sino que aumenta. El tiempo de ejecución al aplicar la directiva es de 111 horas. Como consecuencia, se deben explorar otras líneas de acción y descartar esta. Por mantener una estructura clara, se ilustra la figura que representa este resultado en el apartado dedicado a ello (ver apartado 4.2.3).

3.3.3.3 Procesamiento de los datos dinámicos

Primero, se trata de transformar el código de [1] que usa *pandas*, en un código que trabaje sobre la GPU con *cuDF*. Este proceso ha implicado la reescritura y adaptación de muchas estructuras y operaciones ya que, originalmente, el código ejecutaba estas tareas en CPU con *pandas*. Para superar estas limitaciones, se ha programado el código para que realice el procesamiento en la GPU mediante *cuDF*, lo que permite paralelizar las operaciones y puede contribuir a reducir los tiempos de ejecución de forma significativa.

En primer lugar, es necesario el uso de nuevas bibliotecas. Las bibliotecas implementadas son las siguientes: *cuDF*, *cupy* y *hvplot.cudf*. Estas bibliotecas permitirán ejecutar acciones equivalentes a las ejecutadas en el código base.

En la Figura 3-13 muestra el contenido de la primera fila del fichero *df_raw_dynamic*. En el DF con este mismo nombre se extraerán los datos de los archivos CSV de la carpeta *Dynamic_by_MMSI_month*.

```
# Obtiene la lista de archivos CSV en el directorio
lista_archivos_dynamic = (
    [archivo for archivo in os.listdir('Dynamic_by_MMSI_month') if archivo.endswith('.csv')])
# Lee el primer archivo CSV y lo almacena en un DataFrame
df_raw_dynamic = pd.read_csv(os.path.join('Dynamic_by_MMSI_month', lista_archivos_dynamic[0]))
df_raw_dynamic.head(1)
```

Unnamed: 0	timestamp	mmsi	lat	lon	speed	course	heading	shiptype	
0	101	1674428298	232018697	-10.14435	-34.270712	11.4	26.2	25	ShipType.Cargo

Figura 3-13 Creación de *df_raw_dynamic*

El primer parámetro de análisis inmediato es la cantidad de mensajes transmitidos por cada buque. La Figura 3-14 muestra el código utilizado para almacenar esta información en un DF (*df_longitudes*), registrando el número de mensajes transmitidos y el tipo de buque correspondiente.

Al examinar la distribución de los datos, se puede observar que en el primer cuartil (Q1) se encuentran aproximadamente entre 350 y 500 mensajes, lo que indica que el 25% de los buques han enviado menos de 400 mensajes en el mes aproximadamente. Los valores de los Q1 son los siguientes:

- *ShipType.Cargo* → 404
- *ShipType.Fishing* → 538
- *ShipType.Passenger* → 409
- *ShipType.Tanker* → 352
- *ShipType.Tug* → 512

Estos valores sirven para ir comprobando a lo largo del desarrollo (como haremos con otros datos) que los resultados son similares a los obtenidos en [1]. Así, vemos que estos valores de Q1 coinciden con los obtenidos anteriormente.

Como consecuencia de este análisis, se ha decidido aplicar un filtro de 100 mensajes como umbral mínimo para los experimentos. Esto implica que solo se considerarán aquellos MMSI en los que el número de mensajes transmitidos sea superior a este límite. La decisión de establecer este umbral, además de la experiencia pasada tomada de [1] y queriendo replicar su filtrado, se basa en la observación de que las cadencias de transmisión suelen ser muy superiores a lo esperado, con valores que pueden llegar a 1 mensaje por hora. En consecuencia, una trayectoria con 100 mensajes podría representar un recorrido de 100 horas, lo que podría ser suficiente para definir rutas con cierta precisión.

```
# Inicializa un DataFrame para almacenar las longitudes según el shiptype
data = {'shiptype': [], 'longitud': []}

for MMSI, row in tqdm(df_MMSI_shiptype.iterrows()):
    # Busca el archivo CSV correspondiente al MMSI actual
    csv_file = os.path.join('Dynamic_by_MMSI_month', f"{row['MMSI']}.csv")

    # Verifica si el archivo CSV existe
    if os.path.exists(csv_file):
        df_raw_dynamic = pd.read_csv(csv_file)

        # Calcula la longitud del DataFrame actual y añádela al DataFrame de datos
        shiptype = row['shiptype']
        longitud_df = len(df_raw_dynamic)
        data['shiptype'].append(shiptype)
        data['longitud'].append(longitud_df)

# Crea un DataFrame a partir de los datos recopilados
df_longitudes = pd.DataFrame(data)
```

Figura 3-14 Creación de *df_longitudes*

Una vez creado el DF sobre el que se irán procesando los datos hasta obtener los parámetros deseados para el entrenamiento del modelo, el siguiente paso es la preparación del DF y el procesamiento de los datos.

En [1] se creaban las trayectorias a partir del uso de las bibliotecas *geopandas* y *movingpandas*, algo que en este TFG no es posible debido a la incompatibilidad de estas bibliotecas con *cuDF*. Es por ello que los cálculos de los parámetros de las trayectorias se implementarán ad-hoc en vez de empleando las bibliotecas.

```

# Cargar un archivo CSV específico en cuDF para pruebas
ejemplo_raw = cudf.read_csv(os.path.join('Dynamic_by_MMSI_month', lista_archivos_dynamic[243]))

# Verificar y convertir tipos antes de la fusión
df_MMSI_shiptype = cudf.DataFrame(df_MMSI_shiptype).reset_index() # Asegurar cuDF y resetear índice
df_MMSI_shiptype['MMSI'] = df_MMSI_shiptype['MMSI'].astype('int64')
ejemplo_raw['mmsi'] = ejemplo_raw['mmsi'].astype('int64')

# Realizar merge en cuDF
ejemplo_merged = ejemplo_raw.merge(df_MMSI_shiptype, left_on='mmsi', right_on='MMSI', how='inner')
ejemplo_merged = ejemplo_merged.sort_values(by='timestamp')

# Limpieza de columnas
ejemplo_merged = ejemplo_merged.drop(columns=['MMSI'], errors='ignore')
ejemplo_merged = ejemplo_merged.drop(columns=['shiptype_y'], errors='ignore')
ejemplo_merged = ejemplo_merged.rename(columns={'shiptype_x': 'shiptype'})
ejemplo_merged = ejemplo_merged.dropna(subset=['shiptype']) # Eliminar NaN en shiptype

# Convertir timestamp a formato datetime en cuDF
ejemplo_merged['timestamp'] = cudf.to_datetime(ejemplo_merged['timestamp'], unit='s')
print(ejemplo_merged) # Imprimir resultado para verificar

# Fusionar con el DataFrame de MMSI y shiptype
ejemplo_raw['mmsi'] = ejemplo_raw['mmsi'].astype('int64')
df_MMSI_shiptype['MMSI'] = df_MMSI_shiptype['MMSI'].astype('int64')
ejemplo_merged = ejemplo_raw.merge(df_MMSI_shiptype, left_on='mmsi', right_on='MMSI', how='inner')
ejemplo_merged = ejemplo_merged.sort_values(by='timestamp')
ejemplo_merged = ejemplo_merged.drop(columns=['MMSI'])
ejemplo_merged = ejemplo_merged.drop(columns=['shiptype_y'])
ejemplo_merged = ejemplo_merged.rename(columns={'shiptype_x': 'shiptype'})
ejemplo_merged = ejemplo_merged.dropna(subset=['shiptype'])
ejemplo_merged['timestamp'] = cudf.to_datetime(ejemplo_merged['timestamp'], unit='s')
print(ejemplo_merged)

```

Figura 3-15 Creación del DF *ejemplo_merged*

Mediante el código representado en la Figura 3-15, se crea un DF llamado *ejemplo_merged*, a partir del cual se obtienen los datos dinámicos del archivo que deseemos en *Dynamic_by_MMSI_month*. Para ello, habrá que indicar a qué elemento queremos acceder mediante el código de la línea 2: por ejemplo, con *lista_archivo_dynamic[243]* será el “243” el archivo que coincide en la fila 243 del *index* y que se leerá en *ejemplo_merged*. Esto será usado posteriormente para la representación de los puntos GPS en el instante de la transmisión AIS durante toda la trayectoria para un buque en concreto. Esto permite discernir visualmente entre las trayectorias características de los diferentes barcos. En la Figura 3-16 se representan los puntos mediante la función *hvplot*, función compatible con *cuDF* y que permite una mayor eficiencia computacional, así como disponer de mapas interactivos.

```

# Visualización optimizada en GPU con hvplot
ejemplo_merged.hvplot.points(x='lon', y='lat', geo=True, tiles=True, width=500, height=500, size=5, color='speed', colorbar=True)

```

Figura 3-16 Representación gráfica de los puntos GPS de un buque

En la Figura 3-17, Figura 3-18 y Figura 3-19 se representan gráficamente las trayectorias de un ejemplo de *Tanker*, *Cargo* y *Pesquero*, respectivamente. Nótese que en la derecha se añade una leyenda de colores que indica la velocidad en nudos, cuanto más oscuro, más rápido. En la Figura 3-17 se observa cómo el *Tanker 1* recorre grandes distancias por las aguas de Indonesia a una velocidad constante de aproximadamente 13 nudos, siendo aparentemente características importantes tanto la velocidad como la falta de aceleraciones. En la Figura 3-18 el *Cargo 1* también recorre grandes distancias, esta vez por aguas del norte de Europa y se aprecia que hace muchas más paradas que el *Tanker 1*, una razón puede ser la distribución de la carga por más de un puerto y/o la necesidad de recalar. Es importante recalcar que, en este caso, la velocidad también es aparentemente constante y se mantiene por encima de los 10 nudos. Por último, la Figura 3-19 representa los puntos GPS por los que pasa el *Pesquero 1*. Este tipo de barco normalmente no recalca en diferentes puertos, a no ser que sea extrema necesidad, sino que tiene su puerto base del que sale y al que vuelve después de faenar en alta mar con la carga de pescado. A mayores, la velocidad es menor que en los casos anteriores y siempre se dirige hacia el mismo punto en medio de la mar, a diferencia de otros tipos de barcos, que se dirigen generalmente de un puerto a otro.

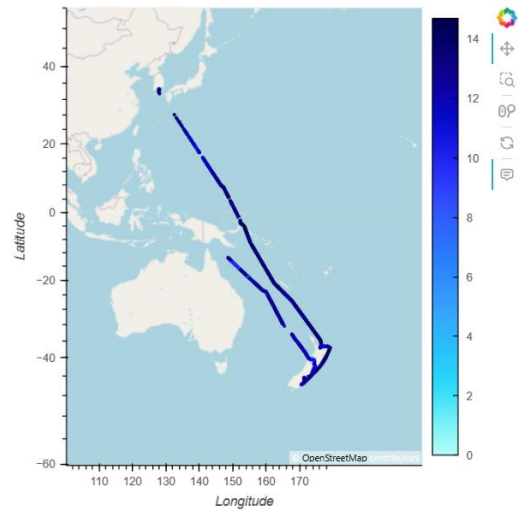


Figura 3-17 Trayectoria del *Tanker 1*

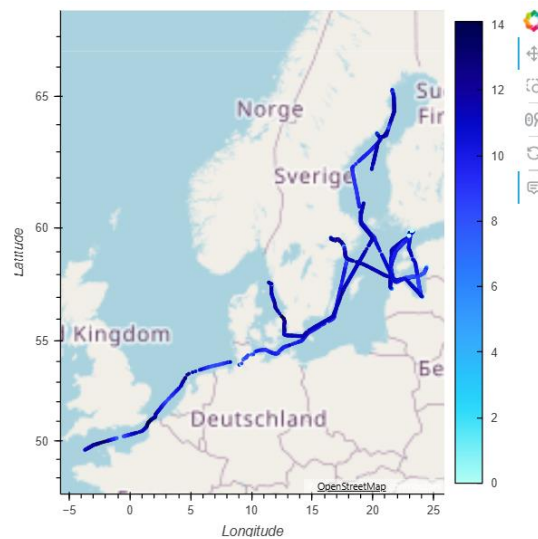


Figura 3-18 Trayectoria del *Cargo 1*

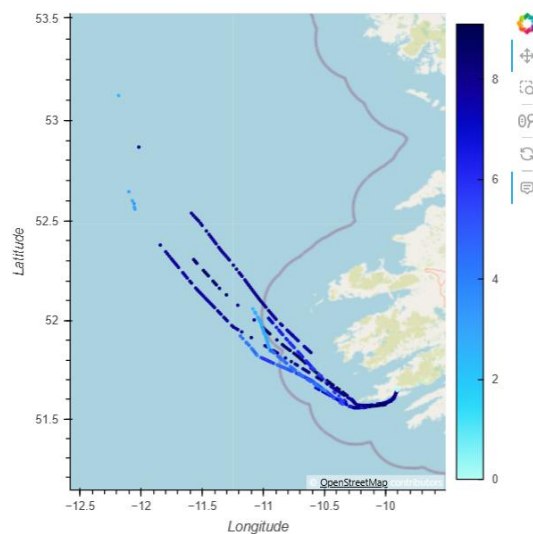


Figura 3-19 Trayectoria del *Pesquero 1*

En [1] se plantea la división de trayectorias, discretizando los momentos en los que el buque está en tránsito de los que está parado. Se extraen parámetros de ambas situaciones por separado. Una vez obtenidas las trayectorias, se plantea el siguiente desafío: discretizar las trayectorias para separarlas y evitar que las paradas contaminen los parámetros obtenidos relacionados con, por ejemplo, la velocidad. De esta manera, se obtendrán parámetros característicos de las trayectorias más realistas y sin estar modificados por todos los mensajes emitidos estando parados. Adicionalmente, se obtendrán también parámetros que definan estas paradas aumentando el número de parámetros que alimentarán al modelo.

Se define trayectoria como el conjunto de mensajes ordenados en el tiempo que un mismo buque transmite entre dos paradas. Las paradas se considerarán como tal cuando una serie de mensajes consecutivos presente posiciones GPS dentro de un área determinada y durante un tiempo mínimo determinado. Finalmente, estos datos de navegación durante un mes se recogen en *month_trajectoryes*.

En [1] se genera el DF *month_trajectoryes* mediante la biblioteca *movingpandas*. Ahora, *cuDF* no es compatible con estas bibliotecas, por lo que debemos implementar la función *calcular_trayectorias*. Aunque en principio podría parecer una limitación no poder recurrir a las bibliotecas *movingpandas* y *geopandas* para crear y discretizar trayectorias mediante sus funciones predeterminadas, esto realmente se traduce en una considerable mejora de la eficiencia, ya que estas bibliotecas tienen un alto consumo computacional por proporcionar muchos otros parámetros y funciones que no serán usados. Al evitar su uso, se elimina la necesidad de representar explícitamente la trayectoria completa de todos los buques. En cambio, se implementa un cálculo directo optimizado para ejecutarse en GPU mediante funciones específicas de *cuDF*, empleando la fórmula de *Haversine* para calcular distancias.

La fórmula de *Haversine* permite calcular la distancia entre dos puntos sobre la superficie de una esfera utilizando sus coordenadas de latitud y longitud. En otras palabras, determina la ruta más corta entre dos ubicaciones en un objeto esférico, como la Tierra.

En la práctica, se utiliza para estimar rutas de vuelo, medir distancias en mapas y en la navegación marítima. También es aplicada en meteorología para definir las trayectorias de tormentas, en la ubicación de equipos de telecomunicaciones y en el seguimiento de satélites. Además, en la navegación GPS ayuda a determinar las rutas más eficientes para los conductores, por lo que será perfecto para aplicarlo en el cálculo de distancia entre la transmisión de dos mensajes [58].

$$d = 2r \cdot \arcsin \left(\sqrt{\sin^2 \left(\frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

Ecuación 3-1 Fórmula de *Haversine*

En la Ecuación 3-1 se representa la fórmula de *Haversine* donde:

- d es la distancia entre los dos puntos.
- r es el radio de la esfera terrestre. Se asume que es perfecta.
- ϕ_1 y ϕ_2 son las latitudes de los puntos 1 y 2 en radianes.
- λ_1 y λ_2 son las longitudes de los puntos 1 y 2 en radianes.

Finalmente, en la Figura 3-20, se muestra la función con la que se ejecuta esta fórmula, definiendo la función *haversine_gpu*, para poder hacer una llamada a la misma posteriormente las veces que sean necesarias.

```
%%time
# Función para calcular distancia entre coordenadas con Haversine en GPU
def haversine_gpu(lat1, lon1, lat2, lon2):
    R = 6371000 # Radio de La Tierra en metros
    phi1 = cp.radians(lat1)
    phi2 = cp.radians(lat2)
    delta_phi = cp.radians(lat2 - lat1)
    delta_lambda = cp.radians(lon2 - lon1)

    a = cp.sin(delta_phi / 2.0) ** 2 + cp.cos(phi1) * cp.cos(phi2) * cp.sin(delta_lambda / 2.0) ** 2
    c = 2 * cp.arctan2(cp.sqrt(a), cp.sqrt(1 - a))
    return R * c
```

Figura 3-20 Implementacion de la fórmula de *Haversine* en *Python*

El cálculo de la distancia es necesario ya que la discretización de las trayectorias se hace mediante la llamada a una función creada explícitamente y, no mediante bibliotecas que ya nos calculaba esta distancia.

Para una mejor comprensión de la migración de las llamadas a funciones con *movingpandas*, se presenta la Tabla 3-5 en la que se expone el cambio de cada función de dicha biblioteca (en la izquierda) a la función de cálculo manual (en la derecha) de los parámetros necesarios. Estas funciones se definen persiguiendo el objetivo de obtener de ellas los mismos parámetros que antes se obtenían con una simple línea de código, pero ahora con un código realizado ad-hoc ganando eficiencia computacional a cambio.

Función de <i>movingpandas</i> usada en [1]	Función definida sin bibliotecas
<i>mpd.Trajectory(df)</i>	<i>calcular_trayectorias</i>
<i>mpd.TrajectoryStopDetector(df).get_stop_points</i>	<i>detectar_paradas_cudf</i>
<i>mpd.TrajectoryStopDetector(df).get_stop_segments</i>	<i>detectar_segmentos_parada_cudf</i>
<i>mpd.StopSplitter(df).split</i>	<i>dividir_trayectorias_por_paradas</i>

Tabla 3-5 Migración de funciones en *movingpandas* a funciones programadas explícitamente en *cuDF*

Primero, se define la función *calcular_trayectorias*, como se muestra en la Figura 3-21. La función comienza ordenando el DF por *mmsi* y *timestamp*, de modo que cada embarcación se procese en orden cronológico. A continuación, se crea la columna *time_diff*, calculando la diferencia de tiempo entre registros consecutivos dentro de cada grupo definido por *mmsi*, y esa diferencia se convierte a un tipo entero de 64 bits. Luego se asignan las columnas *lat_shift* y *lon_shift*, que desplazan la latitud y la longitud del mismo grupo para que cada fila tenga acceso a la posición anterior. Al aplicar la función *shift()*, la primera fila de las columnas *lat_shift* y *lon_shift* contendrá un valor nulo ya que no existe ninguna latitud o longitud anterior. Para evitar futuros fallos debido a estos valores *NaN* (*Not a Number*), se aplica *dropna()* que elimina los valores nulos del DF. Seguidamente, se convierten las columnas *lat*, *lon*, *lat_shift* y *lon_shift* a arrays de *CuPy* para calcular la distancia entre puntos consecutivos mediante la fórmula de *Haversine*, cuyo resultado se almacena en la columna *distance* y se transfiere de la GPU a la memoria principal con el método *.get()*. Acto seguido, se convierte el DF en un objeto de *cuDF* para continuar trabajando con datos acelerados en GPU y se crea la columna *is_stop*, que se activa cuando la distancia es menor de 50 metros y el tiempo transcurrido supera una hora. Esta columna devuelve únicamente 0 y 1 si se cumple que la distancia entre dos mensajes consecutivos sea menor de 50 metros y el tiempo transcurrido mayor a 1 hora (3600 segundos). Por último, se genera la columna *trajectory_id* como una suma acumulativa de *is_stop*, asignando un identificador de trayectoria a cada segmento donde se detecta una parada. Finalmente, la función devuelve el DF con las columnas relevantes para el análisis de trayectorias.

```
def calcular_trayectorias(df):
    df = df.sort_values(by=['mmsi', 'timestamp']) # Ordenar datos en GPU

    df['time_diff'] = df['timestamp'].diff().astype('int64') # Calcular el tiempo entre puntos (en segundos)

    # Calcular distancia entre puntos consecutivos usando CuPy
    df['lat_shift'] = df.groupby('mmsi')['lat'].shift()
    df['lon_shift'] = df.groupby('mmsi')['lon'].shift()
    df = df.dropna()

    lat1 = cp.asarray(df['lat'].values)
    lon1 = cp.asarray(df['lon'].values)
    lat2 = cp.asarray(df['lat_shift'].values)
    lon2 = cp.asarray(df['lon_shift'].values)
    df['distance'] = haversine_gpu(lat1, lon1, lat2, lon2).get()

    # Crear una nueva columna para detectar paradas
    df['is_stop'] = ((df['distance'] < 50) & (df['time_diff'] > 3600)).astype("int32")
    # Crear identificador acumulativo de trayectorias
    df['trajectory_id'] = df['is_stop'].cumsum()

    return df
```

Figura 3-21 Implementación de la función *calcular_trayectorias*

La función *detectar_paradas_cudf* se muestra en la Figura 3-22. La función comienza, al igual que *calcular_trayectorias*, ordenando el DF por *mmsi* y *timestamp*, lo que garantiza que cada barco se procese en orden cronológico. A continuación, calcula la columna *time_diff* agrupando los datos por *mmsi* y obteniendo la diferencia en la columna *timestamp*, para luego convertir esa diferencia en segundos y así conocer el tiempo transcurrido entre mensajes consecutivos. Seguidamente, filtra los registros donde la distancia es menor de 50 metros y el intervalo temporal supera 3600 segundos, generando un subconjunto de filas que representan posibles puntos de paradas, y finalmente devuelve únicamente las columnas relevantes para el análisis que se establecen en la línea 11: *stop_points* `[[‘mmsi’, ‘timestamp’, ‘lat’, ‘lon’, ‘distance’ y ‘time_diff’]]`.

```
# Función para detectar paradas en cuDF (sin MovingPandas)
def detectar_paradas_cudf(df):
    df = df.sort_values(by=['mmsi', 'timestamp']) # Ordenar datos por barco y tiempo

    # Calcular tiempo entre mensajes
    df['time_diff'] = df.groupby('mmsi')['timestamp'].diff().dt.total_seconds()

    # Filtrar puntos de parada (tiempo > 3600s y distancia < 50m)
    stop_points = df[(df['distance'] < 50) & (df['time_diff'] > 3600)]

    return stop_points[['mmsi', 'timestamp', 'lat', 'lon', 'distance', 'time_diff']]
```

Figura 3-22 Implementación de la función *detectar_paradas_cudf*

En la función *detectar_segmentos_parada_cudf*, cuyo código se muestra en la Figura 3-23, se aprovecha la columna binaria *is_stop* generada en *calcular_trayectorias*, y que el DF *month_trajectoryes* contiene para evitar repeticiones de cálculo y optimizar la eficiencia. Con esta columna y mediante el uso de *shift()* y la suma acumulativa (*cumsum()*) crea una nueva columna llamada *split_id*, que incrementa cada vez que se produce una transición de actividad a parada, es decir, de 0 a 1 en la columna *is_stop*, agrupando así los segmentos de trayectorias. El objetivo de la línea 9 es evitar que sume como nueva parada aquellos segmentos consecutivos en los que esté en parada y, por tanto, en ambos *is_stop* = 1. Por último, filtra únicamente las filas en las que *is_stop* es igual a 1 y devuelve un DF con las columnas más relevantes para el análisis de paradas mediante el uso de una máscara booleana (*mask*). La máscara booleana (*df['is_stop'] == 1*) actúa como un filtro lógico que selecciona únicamente las filas del DF donde la condición se cumple, es decir, donde *is_stop* tiene un valor de 1. Esto permite identificar exclusivamente los segmentos de parada.

Posteriormente, al aplicar `.loc[]`, se utiliza esta máscara booleana para extraer solo los datos relevantes, evitando devolver el DF completo con todos los puntos de trayectoria. En lugar de incluir todas las posiciones registradas, la función se centra únicamente en los registros que cumplen la condición de ser una parada (`is_stop == 1`), devolviendo únicamente las columnas especificadas.

```
# Función optimizada para detectar segmentos de paradas en cuDF
def detectar_segmentos_parada_cudf(df):
    df = df.sort_values(by='timestamp') # Ordenar solo por tiempo

    # Calcular diferencia de tiempo en segundos
    df['time_diff'] = df['timestamp'].diff().fillna(0)

    # Identificar si es una parada (distancia < 50m y tiempo > 3600s)
    df['is_stop'] = ((df['distance'] < 50) & (df['time_diff'] > 3600)).astype("int32")

    # Crear identificador de segmento de parada
    df['split_id'] = ((df['is_stop'].shift(fill_value=0) == 0) & (df['is_stop'] == 1)).cumsum()

    # Filtrar solo los segmentos de parada
    mask = df['is_stop'] == 1
    return df.loc[mask, ['mmsi', 'timestamp', 'lat', 'lon', 'distance', 'time_diff', 'split_id']]
```

Figura 3-23 Implementación de la función *detectar_segmentos_parada_cudf*

Una vez se han obtenido los diferentes segmentos de paradas, la siguiente función denominada *dividir_trayectorias_por_paradas* se basa en la información calculada en las etapas anteriores, donde se obtuvieron las columnas *distance* y *time_diff* para cada registro, así como la organización cronológica de los datos por *mmsi*. Primero, define *split_id* marcando con un 1 aquellas filas en las que la distancia entre puntos consecutivos es menor a 50 metros y el tiempo transcurrido supera 1 hora (3600 segundos), y con un 0 en caso contrario. Luego, desplaza esa columna para comparar el valor actual con el anterior y así asegurarse de que solo se detecte el inicio de una parada (es decir, la transición de 0 a 1). Finalmente, convierte esa marca binaria en un identificador de segmento acumulativo mediante la suma progresiva (*cumsum()*), lo que segmenta cada trayectoria en distintos tramos separados por las paradas. El DF resultante incluye columnas como *mmsi*, *timestamp*, *lat*, *lon*, *speed*, *distance*, *time_diff* y el propio *split_id*, que indica a qué tramo pertenece cada punto, siguiendo la misma lógica de detección de paradas utilizada en las funciones anteriores. Lo importante en esta función es la obtención de los parámetros por trayectoria, definida por *split_id*. En la Figura 3-24 se puede ver el código que define esta función.

```
# Función optimizada para dividir trayectorias en segmentos separados por paradas
def dividir_trayectorias_por_paradas(df):
    df['split_id'] = ((df['distance'] < 50) & (df['time_diff'] > 3600)).astype("int32")

    # Asegurar que la segmentación se actualiza solo cuando cambia de 0 → 1
    df['split_id'] = (df['split_id'].shift(fill_value=0) == 0) & (df['split_id'] == 1)
    df['split_id'] = df['split_id'].astype("int32").cumsum()

    return df[['mmsi', 'timestamp', 'lat', 'lon', 'speed', 'distance', 'time_diff', 'split_id', ]]
month_separated_trajectories = dividir_trayectorias_por_paradas(month_trajectories)
print(month_separated_trajectories)
```

Figura 3-24 Implementación de la función *dividir_trayectorias_por_paradas*

En la Figura 3-25 se representa la creación del *df_merged_dynamic*. Primero, se realiza un *merge* entre *df_raw_dynamic* y *df_MMSI_shiptype*, relacionando las columnas *mmsi* (del DF izquierdo) y *MMSI* (del derecho) con un *inner join*, de modo que solo se conservan las filas presentes en ambos.

A continuación, se eliminan las columnas duplicadas que surgen del proceso de fusión (*MMSI* y *shiptype_y*) y se renombra *shiptype_x* a *shiptype* para unificar la nomenclatura. Seguidamente, se descartan las filas que tengan valores nulos en la columna *shiptype*, y finalmente se ordena el DF resultante por *timestamp* antes de imprimirlo. Sobre este nuevo DF se harán posteriormente todas las operaciones. En su creación se utilizan funciones de *pandas* ya que no consume demasiado tiempo. Adicionalmente, se añade una línea que convierte *df_merged_dynamic* de un DF de *pandas* a un DF de *cuDF*.

```
# Realiza el merge basado en las columnas 'mmsi' y 'MMSI'
df_merged_dynamic = pd.merge(df_raw_dynamic, df_MMSI_shiptype, left_on='mmsi', right_on='MMSI',
                             how='inner')
# Elimina la columna 'MMSI' duplicada
df_merged_dynamic.drop(columns=['MMSI'], inplace=True)
# Elimina la columna 'shiptype_y'
df_merged_dynamic.drop(columns=['shiptype_y'], inplace=True)
# Cambia el nombre de 'ship_type_x' a 'shiptype'
df_merged_dynamic.rename(columns={'shiptype_x': 'shiptype'}, inplace=True)
# Elimina filas con valores NaN en la columna 'shiptype'
df_merged_dynamic.dropna(subset=['shiptype'], inplace=True)
# Ordena por timestamp
df_merged_dynamic = df_merged_dynamic.sort_values(by='timestamp')

print(df_merged_dynamic)
```

Figura 3-25 Creación del DF *df_merged_dynamic*

3.3.3.4 Primera iteración

En la primera iteración, se obtendrán los mismos parámetros que se obtuvieron en la cuarta iteración de [1], obviando las iteraciones intermedias. La obtención de parámetros de trayectorias presentó múltiples desafíos que se abordaron en los apartados previos. En primer lugar, se segmentaron los datos en función del MMSI, permitiendo generar un DF que contiene la información de posiciones, así como los parámetros de dirección y rumbo de una misma embarcación. Posteriormente, se identificaron y separaron las trayectorias individuales de cada buque, empleando en lugar de las bibliotecas descritas en [1] las funciones descritas anteriormente. Finalmente, tras la extracción de los parámetros de cada trayectoria de manera individual, se procedió a calcular valores estadísticos agregados.

Por ejemplo, para determinar la velocidad media de un MMSI, primero se calcula la velocidad media de cada trayectoria de forma independiente y, posteriormente, se obtiene el promedio de todas ellas. Aunque a priori esta metodología podría no parecer significativamente distinta de un cálculo directo, la segmentación de las trayectorias permite filtrar registros en los que la embarcación se encuentra detenida (velocidad cercana a cero), ya sea amarrada o fondeada, evitando así que estos valores distorsionen los resultados. Un enfoque similar se aplicará al análisis de los estados de parada, que será tratado en el siguiente apartado.

En el desarrollo de este TFG, se trata de replicar los cálculos y la extracción de los parámetros de la iteración 4 en [1], para contrastar la mejora del tiempo de ejecución además del peso que tiene cada parámetro en la predicción. La definición de los parámetros extraídos se resume en [1], recibiendo en este TFG el mismo nombre y significado, más adelante se describirán los parámetros dinámicos extraídos.

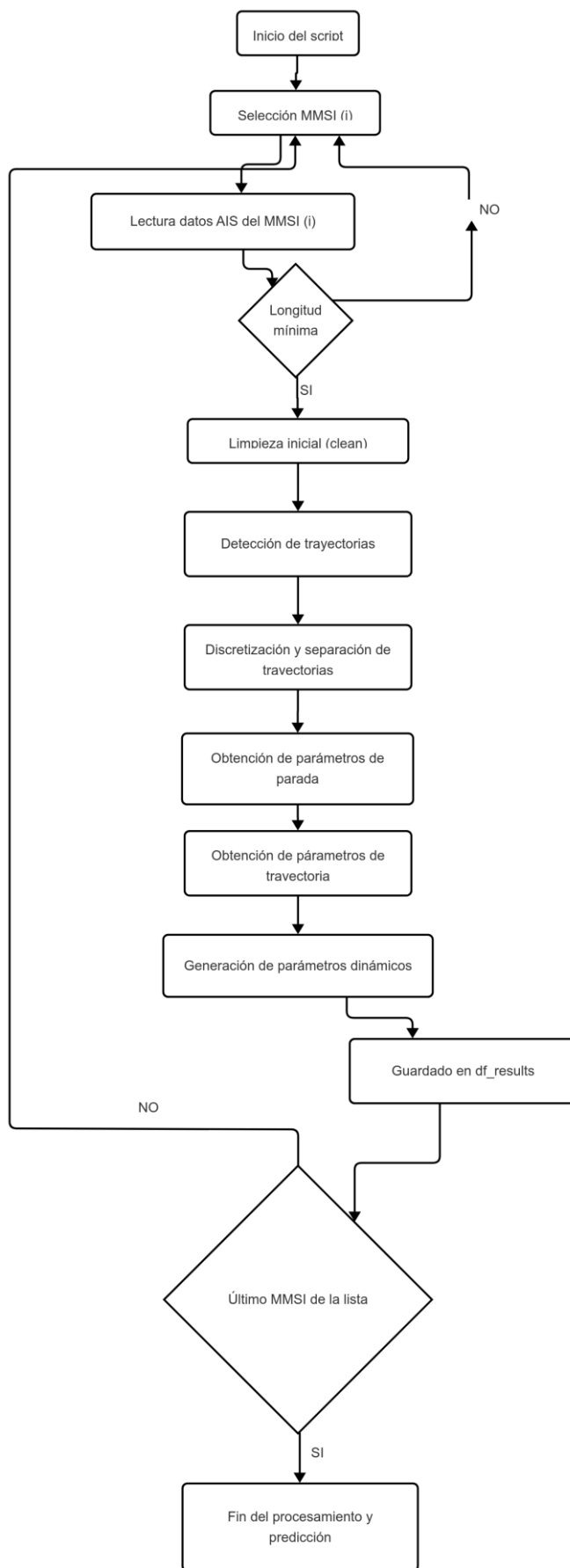


Figura 3-26 Diagrama de flujo del bucle de extracción de parámetros

Para facilitar la comprensión del proceso seguido en el código, se explica el funcionamiento del bucle y sus pasos secuenciales mediante un diagrama de flujo (ver Figura 3-26). En él, se ve la secuencia de llamadas a cada función. Estas funciones se explicarán de manera cronológica según aparecen en el código manteniendo la cohesión y la relación entre ellas. Este bucle también incluye condiciones para asegurar que todo buque que no se lea correctamente o cuya longitud de mensajes no sea suficiente o no se asocie a ningún tipo de barco, se descarte, evitando problemas, al tiempo que se cuantifican para ser conscientes de cuántos buques se descartan.

El bucle comienza por la lectura de un MMSI de *df_MMSI_shiptype*, comprueba si este existe en la carpeta que incluye un archivo por MMSI, en caso de existir comprueba que la longitud sea mayor que 100 mensajes y que tenga un *shiptype* asociado. En caso de cumplir todas las condiciones, pasa a las llamadas de todas las funciones.

Primeramente, se define la función *clean*, representada en la Figura 3-27. La función *clean(df)* tiene como objetivo depurar el DF de columnas innecesarias, estandarizar el formato de la columna tiempo, corregir problemas de duplicados, filtrar valores nulos y normalizar los datos en un DF, ya sea de *pandas* o de *cuDF*.

En esta función se ordenan los registros según la columna *timestamp*, con el fin de asegurar una disposición cronológica de la información y de facilitar análisis temporales o la aplicación de operaciones que dependan del orden del tiempo. Si el conjunto de datos presenta duplicados en las columnas *shiptype_x* y *shiptype_y*, se unifican estos valores en una sola columna (*shiptype*) y se descarta la redundante. También se eliminan las filas con valores nulos (*NaN* o *None*) para mantener la consistencia de la información. Por último, se revisan las columnas de *lat* y *lon* y se descartan los valores que excedan los rangos válidos, sustituyéndolos por *None*. La latitud se restringe al rango $[-90, 90]$ y la longitud al rango $[-180, 180]$. De esta forma, la función devuelve un DF limpio y listo para usos posteriores, especialmente para análisis geoespaciales y temporales.

```
def clean(df):
    df = df.sort_values(by='timestamp') # Ordenar por timestamp

    if 'shiptype_x' in df.columns and 'shiptype_y' in df.columns:
        df = df.rename(columns={'shiptype_x': 'shiptype'}).drop(columns=['shiptype_y'])

    df = df.dropna()
    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), None) # Limitar latitudes
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), None) # Limitar longitudes

    return df
```

Figura 3-27 Implementación de la función *clean*

Una vez limpiado el DF, se le aplican las funciones de detección y discretización de trayectorias según el código mostrado en la Figura 3-28 que aplica las tres funciones descritas anteriormente y utilizadas para la preparación de los datos para la extracción de los parámetros. Estas funciones se aplican sobre *df_merged_dynamic*: primero, *calcular_trayectorias* genera la información de distancias y diferencias de tiempo, luego *detectar_segmentos_parada_cudf* identifica los segmentos en los que se produce una parada y, finalmente, *dividir_trayectorias_por_paradas* separa cada trayectoria según esos puntos de detención. Una vez generados los resultados en *month_trajectories*, *month_stops* y *month_separated_trajectories*, se exportan a archivos CSV para su posterior análisis o procesamiento.

```
# Aplicar funciones en cuDF
month_trajectories = calcular_trayectorias(df_merged_dynamic)
month_stops = detectar_segmentos_parada_cudf(month_trajectories)
month_separated_trajectories = dividir_trayectorias_por_paradas(month_trajectories)
```

Figura 3-28 Aplicación de las funciones de detección y discretización de trayectorias

Tras la discretización de las trayectorias y la obtención de ciertos parámetros agrupados por trayectoria, se pasa al cálculo de los parámetros generales para todo el mes. Este cálculo se realiza mediante las funciones que se describen a continuación.

La función *calcular_parametros_trayectoria* (ver Figura 3-29) tiene como objetivo extraer, a partir de un conjunto de datos de trayectorias (almacenado en *month_separated_trajectories*), una serie de métricas clave que describan cada trayectoria identificada por la columna *split_id*. Además de la distancia total, el tiempo total y las velocidades (media y máxima), la función también calcula la aceleración y desaceleración medias, así como la distribución de la velocidad en distintos rangos. Esto permite caracterizar el comportamiento dinámico de cada trayectoria de manera completa.

A continuación, se describe el código paso a paso:

1. Definición de columnas base del DF de salida. Se declara la lista *traj_parameters*, que contiene los nombres de las columnas que formarán parte del DF de resultados. Entre ellas se incluyen *split_id*, *traj_dist*, *traj_time*, *traj_mean_speed*, *traj_max_speed*, *traj_mean_acc* y *traj_mean_dec*, que representan la distancia total, el tiempo total, la velocidad media, la velocidad máxima y las aceleraciones media y media negativa, respectivamente.
2. Definición de los rangos de velocidad. En *speed_ranges_list* se establecen varios conjuntos de límites para la velocidad. Por ejemplo: [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 10, 15, 20, 300]. Cada lista interna define intervalos de velocidad para los que se calculará la proporción de datos que caen en cada tramo (por ejemplo, de 0 a 0.5, de 0.5 a 1, de 1 a 1.5, etc.).
3. Generación de columnas para los rangos de velocidad. Para cada conjunto de rangos (cada elemento de *speed_ranges_list*), el código recorre los intervalos y genera nuevas columnas en *traj_parameters*, con nombres como *traj_percent_range1_0-0.5*, *traj_percent_range1_0.5-1*, etc. De este modo, se añade una columna por cada intervalo de cada conjunto de rangos.
4. Creación del DF de resultados. Se inicializa un DF de *cuDF* llamado *df_traj_parameters* con las columnas definidas en *traj_parameters*. Aquí se almacenarán los resultados de los cálculos.
5. Agrupación por *split_id*. Se agrupa el DF *month_separated_trajectories* por la columna *split_id*, de modo que cada grupo represente una trayectoria distinta. A continuación, se asigna a la columna *split_id* del DF de resultados el valor de *split_id* correspondiente a cada grupo (tomando el primero de cada grupo).
6. Cálculo de la distancia total por trayectoria. Para cada grupo (trayectoria), se hace la suma de la columna *distance*, lo que da como resultado la distancia recorrida total para cada *split_id* y se almacena en *traj_dist*.
7. Cálculo del tiempo total en segundos. Se resta el *timestamp* de cada fila al de la fila anterior. Estas diferencias, convertidas a segundos, se rellenan con 0 en el primer valor de cada trayectoria. Luego se agrupa por *split_id* y se suman las diferencias, obteniendo el tiempo total (*traj_time*) de cada trayectoria.
8. Cálculo de velocidad media y velocidad máxima. A partir de la columna *speed*, se obtiene la media (*traj_mean_speed*) y el valor máximo (*traj_max_speed*) para cada grupo, lo cual describe la velocidad típica y la velocidad punta en cada trayectoria.
9. Cálculo de aceleraciones y desaceleraciones. Se utiliza la diferencia de la columna *speed* (con *diff()*) para estimar la aceleración en cada punto de la trayectoria. Posteriormente, se calcula la media de dichas diferencias para estimar la aceleración media (*traj_mean_acc*) y la mínima para representar la desaceleración (*traj_mean_dec*).
10. Distribución de velocidades en distintos rangos. Para cada conjunto de rangos en *speed_ranges_list* y cada intervalo dentro de ese conjunto, se calcula la proporción de filas

(puntos de la trayectoria) cuya velocidad se encuentra dentro del intervalo. El resultado se almacena en las columnas creadas en el paso 3 (por ejemplo, *traj_percent_range1_0-0.5*). Esto permite analizar qué porcentaje de la trayectoria transcurre en cada tramo de velocidad.

11. Retorno del DF con resultados. Finalmente, se devuelve *df_traj_parameters*, que contiene para cada *split_id* todas las métricas y distribuciones de velocidad calculadas. Este DF es la síntesis de la información de las trayectorias y sirve para estudios comparativos, modelado de comportamiento dinámico o cualquier análisis que requiera estadísticas agregadas de velocidad y aceleración.

En resumen, el objetivo general de la función *calcular_parametros_trayectoria* es centralizar en un solo DF los valores característicos de cada trayectoria (distancia, tiempo, velocidades, aceleraciones) y la distribución de la velocidad en varios rangos, con el fin de facilitar el análisis y la comparación de rutas.

```
def calcular_parametros_trayectoria(month_separated_trajectories):
    # Definir columnas del DataFrame de salida
    traj_parameters = ['split_id', 'traj_dist',
                       'traj_time', 'traj_mean_speed', 'traj_max_speed',
                       'traj_mean_acc', 'traj_mean_dec']

    # Agregar columnas de rangos de velocidad
    speed_ranges_list = [
        [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20], [0, 5, 10, 15, 20, 300]
    ]

    for sr_index, speed_ranges in enumerate(speed_ranges_list, start=0):
        for i in range(len(speed_ranges) - 1):
            col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
            traj_parameters.append(col_name)

    df_traj_parameters = cudf.DataFrame(columns=traj_parameters) # Crear un DataFrame vacío en cuDF
    grouped = month_separated_trajectories.groupby('split_id') # Agrupar por trayectorias
    df_traj_parameters['split_id'] = grouped['split_id'].first()

    # Calcular distancia total y tiempo entre mensajes en segundos
    df_traj_parameters['traj_dist'] = grouped['distance'].sum()
    df_traj_parameters['traj_time'] = grouped['timestamp'].diff().astype("int64").fillna(0) / 1e9
    # Cálculo de velocidades
    df_traj_parameters['traj_mean_speed'] = grouped['speed'].mean()
    df_traj_parameters['traj_max_speed'] = grouped['speed'].max()
    # Para cada conjunto de rangos de velocidad, crear una columna con el porcentaje de datos que caen en ese tramo
    for sr_index, speed_ranges in enumerate(speed_ranges_list, start=0):
        for i in range(len(speed_ranges) - 1):
            col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
            df_traj_parameters[col_name] = grouped.apply(
                lambda g: (
                    ((g['speed'] >= speed_ranges[i]) & (g['speed'] < speed_ranges[i+1])).sum()
                ) / len(g)
            ) # La variable g, hace referencia a subconjunto del DataFrame que comparte el mismo valor de 'split_id'
    df_traj_parameters['traj_mean_acc'] = grouped['speed'].diff().mean() # Calcular aceleraciones y deceleraciones en GPU
    df_traj_parameters['traj_mean_dec'] = grouped['speed'].diff().min()
    return df_traj_parameters
```

Figura 3-29 Implementación de la función *calcular_parametros_trayectoria*

La función *calcular_parametros_paradas(month_stops)*, detallada en la Figura 3-30, se encarga de agrupar la información de paradas por cada embarcación (identificada mediante la columna *mmsi*) y generar parámetros estadísticos que describan dichas paradas. Esto incluye valores máximos, mínimos y promedios de latitudes, longitudes y diferencias de tiempo, así como el número total de paradas y un rango latitudinal aproximado. El resultado es un conjunto de datos en el que cada fila corresponde a un *mmsi* y cada columna resume métricas clave sobre las paradas de esa embarcación. A continuación, se describe paso a paso el código:

1. Agrupación por *mmsi* y cálculo de estadísticas. El DF de entrada, *month_stops*, se agrupa por la columna *mmsi* y se aplican funciones de agregación como *max*, *min* y *mean* a diversas columnas

(por ejemplo, *stop_max_lat*, *stop_min_lat*, *time_diff_max*, etc.). Esto produce un nuevo DF (almacenado en la variable *grouped*) con estadísticas agregadas, tales como la latitud máxima y mínima de las paradas o la diferencia de tiempo promedio.

2. Renombrado de columnas. Tras la agregación, las columnas suelen quedar con un nombre jerárquico (por ejemplo, *stop_max_lat*, *max*). Para simplificar, se reasignan nombres planos y descriptivos, como *stop_max_lat*, *stop_min_lat* o *stop_mean_lat*.
3. Cálculo de *stop_dist_max*. Se crea la columna *stop_dist_max* como la diferencia entre *stop_max_lat* y *stop_min_lat*, reflejando el rango latitudinal de las paradas de cada *mmsi* (aunque no es una distancia geográfica exacta, sí sirve como aproximación).
4. Cálculo de *stop_N*. De nuevo, se agrupa *month_stops* por *mmsi* y se emplea el método *size()* para contar cuántas filas (paradas) hay por embarcación. Ese valor se añade al DF *grouped* en la columna *stop_N*, indicando cuántas paradas se han registrado para cada *mmsi*.
5. Inserción y orden de columnas. Con *grouped.insert(0, "stop_N", grouped.pop("stop_N"))*, se coloca la columna *stop_N* en la primera posición, facilitando su lectura.
6. Retorno del DF final. La función devuelve el DF *grouped*, donde cada fila corresponde a un *mmsi* y se incluyen los valores agregados (máximos, mínimos, medias, contabilización de paradas y rango latitudinal). Este resultado sirve para el análisis posterior de las paradas de cada embarcación.

Para evitar que la función *detectar_segmentos_parada_cudf* devuelva un DF vacío en los casos en que un buque no cumpla en ningún momento la condición de parada establecida, se han implementado ajustes en el procesamiento de los datos. En lugar de eliminar completamente estos registros, se asignan valores de parada igual a cero en los casos en los que el buque no realiza ninguna detención a lo largo del mes. De este modo, aunque no se calculen parámetros de parada, sí se conservan y procesan el resto de los parámetros dinámicos, evitando la eliminación innecesaria de filas y asegurando la continuidad del análisis.

```
def calcular_parametros_paradas(month_stops):
    if month_stops.shape[0] == 0:
        print("⚠ No hay datos en df para calcular parámetros de paradas.")
        ## Crear un DataFrame con una fila llena de ceros
        df_empty = cudf.DataFrame({
            'mmsi': [0],
            'stop_max_lat': [0], 'stop_min_lat': [0], 'stop_mean_lat': [0],
            'stop_max_lon': [0], 'stop_min_lon': [0], 'stop_mean_lon': [0],
            'N_stops': [0],
            'stop_time_max': [0], 'stop_time_min': [0], 'stop_time_mean': [0],
            'stop_dist_max': [0], 'stop_dist_min': [0], 'stop_dist_mean': [0]
        })
        return df_empty # Retornar DataFrame con ceros
    # Agrupar por MMSI y calcular valores agregados
    grouped = month_stops.groupby('mmsi').agg({
        'lat': ['max', 'min', 'mean'],
        'lon': ['max', 'min', 'mean'],
        'split_id': 'count',
        'time_diff': ['max', 'min', 'mean']
    }).reset_index()
    # Ajustar nombres de columnas
    grouped.columns = ['mmsi', 'stop_max_lat', 'stop_min_lat', 'stop_mean_lat',
                       'stop_max_lon', 'stop_min_lon', 'stop_mean_lon',
                       'N_stops', 'stop_time_max', 'stop_time_min', 'stop_time_mean']
    # **Cálculo de distancias de paradas sin usar apply_grouped**
    stop_distances = month_stops.copy()
    stop_distances['stop_dist'] = (stop_distances.groupby(['mmsi', 'split_id'])['lat']
                                   .diff().abs() +
                                   stop_distances.groupby(['mmsi', 'split_id'])['lon']
                                   .diff().abs()).fillna(0)
    # Obtener las estadísticas de distancia (máxima, mínima y media)
    stop_dist_stats = stop_distances.groupby('mmsi').agg({
        'stop_dist': ['max', 'min', 'mean']
    }).reset_index()
    stop_dist_stats.columns = ['mmsi', 'stop_dist_max', 'stop_dist_min', 'stop_dist_mean'] # Renombrar columnas correctamente
    grouped = grouped.merge(stop_dist_stats, on='mmsi', how='left') # Fusionar resultados finales
    return grouped.fillna(0)
```

Figura 3-30 Implementación de la función *calcular_parametros_paradas*

Finalmente, se programa la función que permite calcular los parámetros dinámicos para la predicción del tipo de buque. Dicha función tendrá como parámetros de entrada el resultado de la ejecución de las funciones presentadas anteriormente. *Calcular_parametros_cinematicos* (*df_traj_parameters*, *month_trajectories*, *df_month_stops*) es el nombre que recibe esta función que tiene como objetivo extraer un conjunto de métricas cinemáticas y espaciales que describen el comportamiento de una embarcación a lo largo de un mes, a partir de datos de trayectoria y de paradas. Para ello, calcula valores estadísticos sobre la distribución espacial (latitudes y longitudes máximas, mínimas y medias), distancias recorridas, tiempos de trayectorias, velocidades y aceleraciones, así como la distribución de velocidades en distintos rangos. El resultado es un DF consolidado que incluye 62 parámetros dinámicos en el que cada fila representa un *mmsi* con todas sus métricas asociadas donde la mayor parte de las columnas son porcentajes en unos rangos de velocidad determinados. Más adelante, se reducirán estas columnas para quedarnos únicamente con aquellas que más peso tengan en la predicción para mejorar los tiempos de ejecución.

Anteriormente, se han calculado los valores estadísticos de estos parámetros por trayectoria. En este caso, se aplican esos cálculos estadísticos sobre los propios valores estadísticos obtenidos previamente, consiguiendo así la media de esos valores en la mayoría de los casos (por ejemplo: *traj_mean_speed_mean* hace referencia a la media de todas las velocidades medias de las trayectorias para un *mmsi*). No obstante, no ha sido materia de estudio en este apartado la evaluación del cálculo de los diferentes parámetros, sino que se han replicado los cálculos de [1].

1. Definición de los parámetros a calcular. Se declara la lista *parameters*, que contiene los nombres de todas las métricas que se van a calcular y almacenar en el DF de salida. Estas métricas incluyen:
 - Identificadores clave (*mmsi*, *N_Trajectories*).
 - Estadísticas geográficas (*month_max_lat*, *month_min_lat*, *month_mean_lat*).
 - Estadísticas de distancias recorridas (*traj_dist_max*, *traj_dist_min*, *traj_dist_mean*).
 - Parámetros temporales (*traj_time_mean*).
 - Velocidades promedio y máximas (*traj_mean_speed_mean*, *month_mean_speed*, *traj_max_speed_max*, *month_max_speed*).
 - Parámetros de aceleración (*traj_mean_acc_mean*, *traj_mean_dec_mean*).
 - Distribución de velocidad en distintos rangos (*traj_percent_0_0dec5_mean*, *traj_percent_5_10_mean*, etc.).
 - Estadísticas de paradas (*N_stops*, *stop_dist_max*, *stop_time_max*, etc.).
2. Inicialización del DF de salida. Se crea un DF vacío de *cuDF* con las columnas definidas en *parameters*, optimizado para su procesamiento en GPU.
3. Cálculo de parámetros relacionados con las paradas. Se obtienen estadísticas a partir del DF *df_month_stops*, tales como el número total de paradas (*N_stops*), las distancias máxima, mínima y media entre paradas (*stop_dist_max*, *stop_dist_min*, *stop_dist_mean*), y los tiempos máximos, mínimos y medios de las paradas (*stop_time_max*, *stop_time_min*, *stop_time_mean*). En este punto es importante destacar que hay buques que no realizan paradas. Para estos, en la función anterior, se disponen todas las variables a 0 con el objetivo de que no devuelva un DF vacío y la ejecución del bucle se interrumpa.
4. Cálculo del número de trayectorias y asignación del *mmsi*. Se extrae el número máximo de trayectorias identificadas (*N_Trajectories*) desde *df_traj_parameters*, y se obtiene el *mmsi* de la embarcación a partir de *month_trajectories*. Es importante asignar el *mmsi* correcto a cada conjunto de parámetros para que, posteriormente, en el *merge* se obtengan los parámetros

característicos para un buque en concreto y así poder relacionarlo con el tipo de buque al que pertenece.

5. Cálculo de parámetros geográficos. Se determinan los valores máximos, mínimos y medios de las coordenadas latitud y longitud (*month_max_lat*, *month_min_lat*, *month_mean_lat*, *month_max_lon*, etc.), así como las diferencias máximas en latitud y longitud (*month_max_dif_lat*, *month_max_dif_lon*), lo que permite conocer la dispersión geográfica de la trayectoria.
6. Cálculo de distancias recorridas. Se mide la cantidad total de registros en *month_trajectories* como una aproximación de la distancia recorrida (*month_dist*). Además, se extraen las distancias máxima, mínima y media de cada trayectoria a partir de *df_traj_parameters*.
7. Cálculo de tiempos y velocidades. Se obtiene el tiempo promedio de las trayectorias (*traj_time_mean*), así como la velocidad media y máxima tanto en las trayectorias (*traj_mean_speed_mean*, *traj_max_speed_max*) como en la serie completa de datos del mes (*month_mean_speed*, *month_max_speed*).
8. Distribución de velocidades en distintos rangos. Se calculan las proporciones de datos que caen en distintos rangos de velocidad. Esto se hace para varios conjuntos de intervalos predefinidos (*Range 1*, *Range 2*, *Range 3* y *Range 4*), permitiendo caracterizar el comportamiento dinámico de la embarcación a diferentes escalas.
9. Cálculo de aceleraciones y desaceleraciones. Se calculan la aceleración media (*traj_mean_acc_mean*) y la desaceleración media (*traj_mean_dec_mean*) a partir de los datos de *df_traj_parameters*.
10. Almacenamiento de los resultados en un DF. Se crea un nuevo DF *df_parameters_temp* en el que se insertan los valores calculados en un formato estructurado, de manera que cada fila representa un único *mmsi* con todas sus métricas cinemáticas y espaciales asociadas. La función devuelve *df_parameters_temp*, el cual contiene el conjunto de las características cinemáticas y espaciales de la embarcación durante el mes.

Este procesamiento permite sintetizar información relevante sobre el comportamiento dinámico de las embarcaciones, facilitando estudios sobre patrones de navegación y será el usado en el bucle final donde se procesarán todos los buques del fichero *Dynamic_by_MMSI_month*. En la Figura 3-31 se reflejan los parámetros extraídos por cada *mmsi*. Estos parámetros coinciden en nombre y significado a los usados en [1]. Los cálculos realizados por medio de estas tres funciones declaradas se usan en el bucle del Anexo III: para la obtención de los parámetros para todos los buques del fichero *Dynamic_by_MMSI_month* durante el mes de enero de 2023.

```
def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories, df_month_stops):
    # Lista de parámetros a calcular
    parameters = ['mmsi', 'N_Trajectories',
                  'month_max_lat', 'month_min_lat', 'month_mean_lat',
                  'month_max_lon', 'month_min_lon', 'month_mean_lon',
                  'month_max_dif_lat', 'month_max_dif_lon',
                  'traj_dist_max', 'traj_dist_min', 'traj_dist_mean',
                  'month_dist', 'traj_time_mean',
                  'traj_mean_speed_mean', 'traj_max_speed_max', 'month_mean_speed',
                  'month_max_speed', 'traj_mean_acc_mean', 'traj_mean_dec_mean',
                  'N_stops', 'stop_dist_max', 'stop_dist_min', 'stop_dist_mean',
                  'stop_time_max', 'stop_time_min', 'stop_time_mean',
                  # Rangos de velocidad (Range 1)
                  'traj_percent_0_0dec5_mean', 'traj_percent_0dec5_1_mean',
                  'traj_percent_1_1dec5_mean', 'traj_percent_1dec5_2_mean',
                  'traj_percent_2_2dec5_mean', 'traj_percent_2dec5_3_mean',
                  'traj_percent_3_3dec5_mean', 'traj_percent_3dec5_4_mean',
                  'traj_percent_4_4dec5_mean', 'traj_percent_4dec5_5_mean',
                  # Rangos de velocidad (Range 2)
                  'traj_percent_0_1_mean', 'traj_percent_1_2_mean', 'traj_percent_2_3_mean',
                  'traj_percent_3_4_mean', 'traj_percent_4_5_mean', 'traj_percent_5_6_mean',
                  'traj_percent_6_7_mean', 'traj_percent_7_8_mean', 'traj_percent_8_9_mean',
                  'traj_percent_9_10_mean',
                  # Rangos de velocidad (Range 3)
                  'traj_percent_0_2_mean', 'traj_percent_2_4_mean', 'traj_percent_4_6_mean',
                  'traj_percent_6_8_mean', 'traj_percent_8_10_mean', 'traj_percent_10_12_mean',
                  'traj_percent_12_14_mean', 'traj_percent_14_16_mean', 'traj_percent_16_18_mean',
                  'traj_percent_18_20_mean',
                  # Rango de velocidad (Range 4)
                  'traj_percent_0_5_mean', 'traj_percent_5_10_mean', 'traj_percent_10_15_mean',
                  'traj_percent_15_20_mean', 'traj_percent_over20_mean'
                  ]
```

Figura 3-31 Declaración de parámetros dinámicos a extraer en la primera iteración

Los parámetros calculados en este código se organizan en distintos grupos, cada uno aportando información específica sobre las trayectorias geoespaciales analizadas. A continuación, se detallan los parámetros:

Parámetros generales:

N_Trajectories: Número total de trayectorias del mes.

Parámetros geográficos mensuales:

- *month_max_lat*: Latitud máxima del mes.
- *month_min_lat*: Latitud mínima del mes.
- *month_mean_lat*: Latitud media del mes.
- *month_max_lon*: Longitud máxima del mes.
- *month_min_lon*: Longitud mínima del mes.
- *month_mean_lon*: Longitud media del mes.
- *month_max_dif_lat*: Diferencia máxima mensual entre latitudes.
- *month_max_dif_lon*: Diferencia máxima mensual entre longitudes.

Distancias mensuales:

- *traj_dist_max*: Distancia máxima recorrida en una trayectoria del mes.
- *traj_dist_min*: Distancia mínima recorrida en una trayectoria del mes.
- *traj_dist_mean*: Distancia media recorrida en las trayectorias del mes.
- *month_dist*: Distancia total recorrida durante todo el mes.

Parámetros temporales mensuales:

- *traj_time_mean*: Tiempo medio transcurrido en las trayectorias del mes.

Parámetros mensuales de aceleración y deceleración:

- *traj_mean_acc_mean*: Media de la aceleración media de las trayectorias.
- *traj_mean_dec_mean*: Media de la deceleración media de las trayectorias.

Parámetros mensuales de paradas:

- *N_stops*: Número total de paradas realizadas en el mes.
- *stop_dist_max*: Máxima distancia recorrida durante una parada.
- *stop_dist_min*: Mínima distancia recorrida durante una parada.
- *stop_dist_mean*: Distancia media recorrida durante las paradas del mes.
- *stop_time_max*: Tiempo máximo de duración de una parada.
- *stop_time_min*: Tiempo mínimo de duración de una parada.
- *stop_time_mean*: Tiempo medio de duración de las paradas.

Parámetros de velocidad:

- *traj_mean_speed_mean*: Media de la velocidad media de las trayectorias.
- *traj_max_speed_max*: Velocidad máxima alcanzada en las trayectorias.
- *traj_min_speed_max*: Velocidad mínima alcanzada en las trayectorias.
- *month_mean_speed*: Velocidad media general del mes.
- *month_max_speed*: Velocidad máxima mensual.
- *month_min_speed*: Velocidad mínima mensual.
- *traj_percent[rango]_mean*: Indican el porcentaje promedio del tiempo transcurrido en diferentes rangos de velocidades durante las trayectorias del mes.

Esta estructura y el uso directo de cálculos optimizados para GPU con *cuDF* permiten mejorar considerablemente la eficiencia del proceso, reduciendo notablemente los tiempos de ejecución y carga computacional del análisis. Las funciones descritas serán usadas para la extracción de los parámetros de todos los buques del mes de enero de 2023 a través del uso del bucle presente en el código en el Anexo III: .

3.3.3.5 Segunda iteración

Tras obtener y analizar los resultados de la primera iteración, se ha detectado que el número de parámetros calculados y utilizados en la predicción es excesivo. Muchos de estos parámetros tienen una contribución marginal o prácticamente nula en el desempeño del modelo, lo que sugiere posibilidad de la reducción de parámetros sin perder eficacia en la predicción.

Por este motivo, se lleva a cabo una segunda iteración en la que se eliminan aquellos parámetros con menor impacto en la predicción, calculando únicamente aquellos que presentan una relevancia significativa. Esta selección se basa en la gráfica de importancia de los parámetros obtenida en el *Experimento datos globales 2* de la Primera Iteración, representada en la Figura 3-32. El objetivo de esta reducción es doble: por un lado, simplificar el modelo sin comprometer su capacidad predictiva y, por otro, optimizar los tiempos de ejecución del procesamiento. Para que la calidad de la predicción no se vea afectada de manera significativa se establece un umbral de tolerancia, de modo que la eliminación de parámetros no debe reducir la precisión del modelo en más de 0,01 unidades (con una precisión de dos decimales).

En términos de organización, los resultados detallados de esta segunda iteración se presentan en el apartado 4.2. No obstante, para justificar la necesidad de este proceso, se incluye en la Figura 3-32 la gráfica que muestra el peso de los diferentes parámetros dentro del modelo. Esta gráfica corresponde a la predicción realizada con el conjunto de datos *df_global_parameters_1*, el cual integra tanto los

Se lleva a cabo una segunda prueba preliminar en la que se seleccionan más parámetros para eliminar y se llega a la conclusión de que se puede seguir reduciendo los parámetros. Esta vez, en la prueba preliminar se eliminan las columnas de la Figura 3-34.

```
# SEGUNDA PRUEBA PRELIMINAR RANDOM FOREST
# eliminamos mas columnas
columnas_a_eliminar = ['traj_percent_5_6_mean', 'traj_percent_9_10_mean', 'traj_percent_1_2_mean',
                      'traj_percent_2dec5_3_mean', 'traj_percent_14_16_mean', 'traj_time_mean',
                      'traj_percent_8_10_mean', 'traj_percent_10_12_mean',
                      'traj_percent_1_2_mean', 'traj_percent_2dec5_3_mean', 'traj_percent_4dec5_5_mean',
                      'traj_percent_4_5_mean', 'traj_percent_3_4_mean',
                      'traj_percent_2_3_mean', 'traj_percent_0dec5_1_mean',
                      'traj_percent_4_6_mean']

# Eliminar las columnas especificadas, ignorando errores si alguna no existe
df_global_parameters_3_RF = df_global_parameters_2_RF.drop(columns=columnas_a_eliminar, errors='ignore')
```

Figura 3-34 Segunda prueba preliminar para eliminar columnas en la segunda iteración en *Random Forest*

	precision	recall	f1-score	support
0	0.90	0.95	0.92	4571
1	0.89	0.91	0.90	1052
2	0.85	0.84	0.85	570
3	0.91	0.78	0.84	1791
4	0.91	0.90	0.91	1223
accuracy			0.90	9207
macro avg	0.89	0.88	0.88	9207
weighted avg	0.90	0.90	0.90	9207

Figura 3-35 Resultados de la segunda prueba preliminar de *Random Forest*

Una vez se ha comprobado que, al haber eliminado estos parámetros, la calidad no se ve afectada significativamente tal y como se observa en la Figura 3-35, se vuelven a modificar las funciones en el código principal para extraer únicamente los parámetros cinemáticos que deseamos. De esta manera, se establecen los parámetros finales con los que se evalúa la segunda iteración.

```
def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories):
    # Lista de parámetros a calcular
    parameters = ['mmsi',
                  'month_max_lat', 'month_min_lat', 'month_mean_lat',
                  'month_max_lon', 'month_min_lon', 'month_mean_lon',
                  'month_max_dif_lat', 'month_max_dif_lon',
                  'traj_dist_max', 'traj_dist_mean',
                  'month_dist', 'traj_time_mean',
                  'traj_mean_speed_mean', 'traj_max_speed_max', 'month_mean_speed',
                  'month_max_speed', 'traj_mean_dec_mean',
                  # Rangos de velocidad (Range 3)
                  'traj_percent_2_4_mean',
                  'traj_percent_12_14_mean',
                  # Rango de velocidad (Range 4)
                  'traj_percent_10_15_mean'
                  ]
```

Figura 3-36 Declaración de parámetros dinámicos a extraer en la segunda iteración

En la Figura 3-36 se detallan los parámetros que se han definido para su extracción durante la segunda iteración. Como se puede observar, el número de parámetros se ha reducido drásticamente con respecto a los parámetros extraídos en la primera iteración, en concreto, se ha reducido a 20 parámetros si no tenemos en cuenta el *mmsi*, ya que será eliminado para la predicción. Además, las propias funciones de *calcular_parametros_paradas* y *calcular_parametros_trayectoria* se han modificado para calcular únicamente los parámetros que nos interesan para esta iteración, obteniendo así una mayor eficiencia computacional que, definitivamente, es el objetivo tras estos experimentos. Los resultados de esta iteración se muestran y se analizan en el capítulo reservado para ello para mantener la organización de la memoria (ver capítulo 4).

Al aplicar *XGBoost* para la predicción, se observa que la importancia de los distintos parámetros cambia. El motivo se explica posteriormente en la evaluación de los resultados con el objetivo de no

alterar la estructura de la memoria (ver Apartado 4.2.4). Además, la precisión sube a más del 91%. Es por ello que, en la segunda iteración, surgen dos líneas de acción: eliminar las columnas de menor peso para *Random Forest* como ya se ha explicado y eliminar las columnas de menor peso para *XGBoost* paralelamente. De esta manera, nos aseguramos de que se alimente el algoritmo de *XGBoost* con los parámetros que este algoritmo toma como más importantes.

Para seleccionar los parámetros que se eliminarán, se procede de la misma manera: se hace una prueba preliminar en la que se comprueba que la precisión no se ve afectada significativamente (más de un 1%). Por hacer una comparativa directa con *Random Forest*, se van a eliminar el mismo número de parámetros cinemáticos y se mantendrán todos los estáticos como con el algoritmo anterior. A pesar de que en *XGBoost* hay parámetros estáticos que aportan poco peso a la predicción, como su extracción no requiere de mucho tiempo de ejecución, no ganaremos eficiencia computacional al dejar de calcularlos, por tanto, no aporta beneficios al objetivo del TFG.

```
# PRUEBA PRELIMINAR XGBOOST
# eliminamos mas columnas
columnas_a_eliminar = ['stop_time_min', 'traj_time_mean', 'traj_mean_acc_mean',
                      'stop_time_max', 'traj_percent_2_2dec5_mean', 'stop_dist_mean',
                      'traj_percent_7_8_mean', 'traj_percent_1_2_mean',
                      'stop_time_mean', 'traj_percent_1_1dec5_mean', 'month_max_speed',
                      'traj_dist_min', 'to_starboard', 'stop_dist_max', 'traj_percent_0_0dec5_mean',
                      'traj_percent_8_9_mean', 'traj_percent_8_10_mean', 'traj_mean_speed_mean',
                      'N_stops', 'month_dist', 'traj_percent_10_12_mean', 'traj_percent_5_6_mean',
                      'traj_percent_6_8_mean', 'traj_percent_1dec5_2_mean',
                      'traj_percent_5_10_mean', 'traj_percent_0_2_mean', 'traj_percent_9_10_mean',
                      'traj_percent_4_6_mean', 'traj_percent_2_3_mean', 'traj_dist_mean',
                      'traj_max_speed_max', 'traj_percent_4_5_mean', 'N_Trajectories',
                      'month_mean_speed', 'traj_percent_3_4_mean', 'traj_percent_3_3dec5_mean',
                      'traj_percent_6_7_mean', 'traj_percent_0_1_mean',
                      'traj_percent_0dec5_1_mean', 'traj_percent_4_4dec5_mean']

# Eliminar las columnas especificadas, ignorando errores si alguna no existe
df_global_parameters_2_XG = df_global_parameters_1.drop(columns=columnas_a_eliminar, errors='ignore')
```

Figura 3-37 Prueba preliminar para eliminar columnas en la segunda iteración en *XGBoost*

En la Figura 3-37 se observan las columnas eliminadas en la prueba preliminar. Ahora se extraerán 20 parámetros cinemáticos, al igual que para *Random Forest* con la diferencia de que los parámetros no serán los mismos. Una vez eliminadas las columnas, se continúa con la prueba preliminar ejecutando *XGBoost* para el DF *df_global_parameters_2_XG* que contiene los parámetros estáticos (no se ha eliminado ninguno) y los 20 parámetros cinemáticos con mayor importancia para la predicción. En la Figura 3-38 se justifica que no solo no se pierde precisión al calcular menos parámetros, sino que incluso aumenta. La explicación de por qué aumenta la precisión al usar menos parámetros viene proporcionada en [59]. En este libro se profundiza en los conceptos de sobreajuste y en cómo la reducción de la dimensionalidad (es decir, eliminar parámetros o características irrelevantes) contribuye a obtener modelos más robustos y con mejor capacidad de generalización con menos parámetros, el modelo se vuelve menos complejo y, por tanto, más robusto. Usar menos parámetros para la predicción ayuda a evitar problemas derivados de la alta dimensionalidad que pueden dificultar la correcta estimación de las relaciones entre las variables, además de que al trabajar con un subconjunto de parámetros que tienen mayor relevancia, *XGBoost* puede aprovechar mejor la información disponible, focalizando su poder predictivo en las variables que realmente aportan valor al modelo.

Precisión del modelo: 0.9153

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.92	0.95	0.93	3650
1	0.92	0.92	0.92	886
2	0.87	0.87	0.87	466
3	0.90	0.84	0.87	1424
4	0.92	0.92	0.92	939
accuracy			0.92	7365
macro avg	0.91	0.90	0.90	7365
weighted avg	0.92	0.92	0.91	7365

Figura 3-38 Resultados de la prueba preliminar de *XGBoost*

Comprobado que no se pierde precisión en los resultados, se pasará a modificar el código para ganar eficiencia mediante el cálculo de menos parámetros.

3.3.3.6 Optimización del acceso a los datos: eliminación del bucle y carga unificada

Hasta ahora, el procesamiento de datos se ha centrado en la iteración sobre varios archivos independientes, accediendo a la información para cada MMSI mediante un bucle que procesaba cada archivo por separado. Este enfoque generaba un alto coste computacional debido al acceso repetitivo a disco y la fragmentación de los datos, lo que disminuía significativamente la eficiencia del procesamiento. No obstante, era necesario para asegurar la escalabilidad del tamaño de los datos.

Para mejorar el rendimiento, en este experimento se ha implementado un nuevo enfoque: en lugar de leer múltiples archivos, todos los datos han sido fusionados en un único archivo de gran tamaño, el cual se carga directamente en un DF y sobre el que se realizan todas las operaciones de manera global. Esta modificación elimina la necesidad de iterar sobre cada archivo, permitiendo aprovechar las ventajas del procesamiento en GPU y reduciendo considerablemente los tiempos de ejecución.

El propósito principal de esta optimización es disminuir el impacto de las operaciones de entrada/salida y mejorar la eficiencia computacional al permitir que *cuDF* procese directamente un conjunto de datos más amplio, maximizando la paralelización y reduciendo los cuellos de botella generados por el acceso secuencial a múltiples archivos.

Este enfoque implica cargar todos los datos dinámicos de todos los buques en memoria, para poder acceder a ellos de forma simultánea. Existe un límite a este tamaño de datos en memoria, impuesto por la capacidad *hardware* de la GPU utilizada. Si el conjunto de datos es mayor a esa capacidad, la lectura dará error. Además, las operaciones posteriores, que implican generación de nuevas columnas, cálculos entre ellas, generación de estructuras de datos auxiliares, también incrementan el uso de esa memoria, por lo que se pueden dar nuevos errores en tiempo de ejecución. Adicionalmente, la biblioteca *cuDF* (para conseguir paralelización) consume de forma interna memoria GPU. Es por todo ello que ha sido necesario reestructurar, redefinir y eliminar estructuras de datos auxiliares para evitar estos errores en tiempo de ejecución por falta de memoria, que implicaban reiniciar el *kernel* (el proceso que ejecuta el *script* de Python y que mantiene los datos en memoria) y tener que repetir todo el proceso. Debemos remarcar que ha sido posible tener éxito debido al tamaño de los datos dinámicos con los que trabajábamos. En caso de tener datos dinámicos de un rango temporal mayor, o generados con mayor frecuencia, es decir, un tamaño de datos superior, se generarían los errores en tiempo de ejecución de los que hablamos. Es decir, existe un problema de escalabilidad con esta solución.

A continuación, se explican las modificaciones realizadas para poder solventar los problemas de consumo de memoria en GPU.

En primer lugar, se define el DF *df_dynamic* en el que se almacenan los datos de todos los buques. A partir de este punto, se realizan cálculos y operaciones modificando este *df_dynamic*. Así, se evita crear nuevos DF y, por ende, que aumente el uso de la memoria. De esta manera, se sobre escribirá *df_dynamic* depurándolo para obtener los parámetros deseados. La dinámica es la misma que la que se ha seguido en los experimentos anteriores. Se han programado explícitamente las funciones que se han usado también en el resto de los experimentos para poder aplicarlo sobre un DF ordenado por *mmsi* y *timestamp*.

En la función *clean* se modifica los tipos de datos a unos menos precisos, en función de las necesidades de cada variable, con objeto de reducir su peso en la memoria. A continuación, se hace un mapeo para modificar el tipo de datos en la columna *shiptype*. Finalmente, se eliminan los datos no lógicos al igual que se hacía anteriormente. Esta función se muestra en la Figura 3-39.

```

def clean(df):
    df['mmsi'] = df['mmsi'].astype('int32') # ID del buque, ocupa menos espacio
    df['timestamp'] = df['timestamp'].astype('int64') # Mejor formato para cálculos de tiempo
    df['lat'] = df['lat'].astype('float32') # Precisión suficiente para coordenadas
    df['lon'] = df['lon'].astype('float32')
    df['speed'] = df['speed'].astype('float32') # Precisión suficiente para velocidad
    df['course'] = df['course'].astype('float32')
    df['heading'] = df['heading'].astype('int16') # Enteros pequeños ahorran memoria
    # Mapeo de shiptype a valores numéricos
    shiptype_mapping = {
        'ShipType.Cargo': 0,
        'ShipType.Fishing': 1,
        'ShipType.Passenger': 2,
        'ShipType.Tanker': 3,
        'ShipType.Tug': 4,
    }
    # Reemplazar valores con el mapping y convertir a int8
    df['shiptype'] = df['shiptype'].map(shiptype_mapping).fillna(-1).astype('int8')
    df = df.sort_values(by=['mmsi', 'timestamp']) # Ordenar datos en GPU

    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), None) # Limitar Latitudes
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), None) # Limitar Longitudes

    # Eliminamos los MMSI con menos de 100 entradas
    counts = df.groupby('mmsi').size().reset_index(name='count')
    valid_mmsi = counts[counts['count'] >= 100]['mmsi']
    df = df[df['mmsi'].isin(valid_mmsi)]

    df.reset_index(inplace=True)
    df = df.drop('Unnamed: 0', axis=1)
    df = df.drop(columns=['index'])

    return df

```

Figura 3-39 Implementación de la función *clean* para optimizar el acceso a los datos

En esta nueva implementación de la función *calcular_trayectorias*, se ha modificado significativamente el enfoque en comparación con versiones anteriores, optimizando tanto la segmentación de datos como la eficiencia computacional. Uno de los cambios más relevantes es el cálculo de dos DF separados: *df_paradas* y *df_trayectorias*, los cuales distinguen las trayectorias en movimiento de los períodos de detención de los buques. Esto se logra filtrando el conjunto de datos mediante la variable *is_stop*, asignando a *df_paradas* las filas donde *is_stop* == 1 y a *df_trayectorias* aquellas donde *is_stop* == 0. Esta diferenciación evita la mezcla de estados en un mismo análisis y permite un estudio más preciso tanto de la navegación como de los tiempos de espera y fondeo. En estudios anteriores, para separar trayectorias en movimiento y paradas, se utilizaban varias funciones, como *detectar_segmentos_parada_cudf* y *dividir_trayectorias_por_paradas*, lo que implicaba ejecutar múltiples operaciones sobre el DF en diferentes etapas del procesamiento.

Ahora, al integrar ambas segmentaciones en una sola función, se reduce la sobrecarga computacional y se eliminan cálculos redundantes, lo que mejora significativamente la eficiencia del código y evita problemas de consumo de memoria. Otro cambio clave es la nueva estrategia de asignación del identificador de trayectoria, *trajectory_id*. En versiones previas, este identificador se reiniciaba para cada buque, lo que implicaba el uso de bucles o cálculos adicionales sobre el conjunto de datos. Ahora, los *trajectory_id* se acumulan globalmente en todo el DF, eliminando iteraciones innecesarias y reduciendo considerablemente los tiempos de procesamiento.

Además, se ha redefinido la condición para detectar paradas con el objetivo de mejorar la segmentación de trayectorias y minimizar errores en zonas de entrada y salida de puerto. Mientras que anteriormente se ha usado el criterio establecido en [1] para no modificar los cálculos, en esta versión, un buque se considera detenido si su velocidad es inferior a 1 km/h y la distancia acumulada en esa trayectoria es menor a 10 km. Esto evita la detección de múltiples trayectorias erróneas en situaciones en las que el buque está detenido temporalmente o realizando maniobras en puerto.

También se ha incorporado un filtro para eliminar falsas paradas, es decir, aquellas que solo duran un mensaje (una fila en el DF). Para ello, se han implementado las columnas auxiliares *prev* y *next*, que verifican si una fila con *is_stop* == 1 está rodeada por valores de *is_stop* == 0. En estos casos, la parada se reclasifica para evitar que distorsione el análisis de trayectorias reales.

Ahora, la detección de transiciones entre estados permite segmentar con mayor precisión los cambios entre movimiento y parada. De esta manera, los datos de parada y de movimiento no formarán parte de la misma trayectoria asegurando una mayor precisión estadística. En esta nueva versión, se considera que una nueva trayectoria comienza cuando:

- Un buque cambia de $is_stop == 1$ a $is_stop == 0$, es decir, cuando reanuda su movimiento.
- Un buque cambia de $is_stop == 0$ a $is_stop == 1$, indicando el inicio de una trayectoria de parada.
- Se detecta un cambio en el MMSI, lo que señala la aparición de un nuevo buque en el conjunto de datos.

```
# Detectar transiciones de parada a movimiento
df['transitions'] = ((df['is_stop'].shift(1) == 1) & (df['is_stop'] == 0)) | (df['mmsi'].shift(1) != df['mmsi'])

# Contar transiciones acumuladas para segmentar trayectorias
df['trajectory_id'] = df['transitions'].cumsum().astype('int32')

# Calcular la distancia acumulada dentro de cada trayectoria
df['dist_acum'] = df.groupby('trajectory_id')['distance'].cumsum().astype('int32')
df['tiempo_acum'] = df.groupby('trajectory_id')['time_diff'].cumsum().astype('int32')

# Crear un filtro para las trayectorias con dist_acum < DISTANCE_THRESHOLD
low_distance_ids = df.groupby('trajectory_id').tail(1)
low_distance_ids = low_distance_ids[low_distance_ids['dist_acum'] < DISTANCE_THRESHOLD]['trajectory_id'].unique()

# Asignar is_stop = 1 a las filas con esos trajectory_id
df['is_stop'] = df['is_stop'].where(~df['trajectory_id'].isin(low_distance_ids), 1)

# Eliminar valores aislados de is_stop (0,0,0,0,0,1,0,0,0 -> 0)
df['prev'] = df['is_stop'].shift(1).fillna(0)
df['next'] = df['is_stop'].shift(-1).fillna(0)
df['is_stop'] = df['is_stop'].where((df['prev'] == 1) | (df['next'] == 1), 0)

# Detectar transiciones de parada a movimiento
df['transitions'] = ((df['is_stop'].shift(1) == 1) & (df['is_stop'] == 0)) | (df['is_stop'].shift(1) == 0) &
                    (df['is_stop'] == 1) | (df['mmsi'].shift(1) != df['mmsi'])

# Contar transiciones acumuladas para segmentar trayectorias
df['trajectory_id'] = df['transitions'].cumsum().astype('int32')
```

Figura 3-40 Implementación del código para la detección de paradas

En la Figura 3-40 se muestra la nueva implementación de la detección de las trayectorias de los buques, diferenciando entre períodos de movimiento y paradas, además de eliminar falsas detecciones y optimizar la asignación de identificadores de trayectoria.

Con este nuevo enfoque, las trayectorias no solo reflejan los desplazamientos, sino que también incluyen segmentos de tiempo en los que el buque permanece detenido, generando trayectorias de parada. Esto incrementa el número total de trayectorias detectadas, ya que una misma trayectoria se fragmenta en períodos distintos de movimiento y detención, proporcionando un reflejo más detallado del patrón de navegación del buque.

Esta diferenciación permite analizar con mayor precisión el comportamiento de los buques, ofreciendo información detallada sobre tiempos de espera, fondeo o permanencia en puertos. Además, al segmentar correctamente las trayectorias, se minimiza el riesgo de que los periodos de detención sean interpretados erróneamente como parte de una misma trayectoria en movimiento, lo que mejora significativamente la calidad de los datos utilizados en el modelo de predicción.

Las modificaciones en las funciones de cálculo de parámetros se resumen en la agrupación por trayectoria o por la columna *mmsi* para evitar que se calculen los parámetros mezclando barcos. Las funciones *calcular_parametros_trayectoria* y *calcular_parametros_paradas* han sido modificadas para que los cálculos se realicen agrupando los datos por trayectorias, asegurando que cada trayectoria esté correctamente diferenciada por la columna *mmsi*. De esta manera, los parámetros se calculan de forma independiente para cada trayectoria detectada, evitando mezclar información de distintos buques.

Por otro lado, la función *calcular_parametros_cinematicos* también ha sido ajustada pero, en este caso, la agrupación se ha realizado únicamente por la columna *mmsi*, manteniendo la lógica original para el resto de los cálculos. Esta modificación permite que los parámetros cinemáticos reflejen correctamente el comportamiento global de cada buque, sin alterar el procesamiento de las demás métricas.

Por último, el bucle usado antes de cambiar el enfoque de acceso a los datos se resume a las líneas de código representadas en la Figura 3-41. De esta forma, las operaciones se producen sobre *df_dynamic* directamente sin tener que iterar por MMSI. Como se observa, a partir de la función *calcular_trayectorias* se obtienen los DF *df_paradas* y *df_trayectorias* filtrando si para las filas se detecta parada o no, respectivamente. Una vez obtenidos estos DF, se calculan los parámetros de parada y de trayectoria independientemente. Por último, se sustituyen los valores *NaN* con 0, ya que hay para buques para los que no se obtiene ninguna parada o ninguna trayectoria de movimiento. Por tanto, para aprovechar el resto de los parámetros obtenidos, se ponen estos parámetros a 0 y así evitar problemas al alimentar al modelo. Como resultado, se obtiene el DF *parametros_cinematicos* que recoge las 64 mismas columnas que se obtuvieron en la primera iteración.

```
%%time
df_dynamic = cudf.read_csv('dynamic.csv')
df_dynamic.reset_index(inplace=True)
df_dynamic=df_dynamic.drop('Unnamed: 0',axis=1)
df_dynamic = clean(df_dynamic)
df_dynamic['lat_shift'] = df_dynamic.groupby('mmsi')['lat'].shift().astype('float32')
df_dynamic['lon_shift'] = df_dynamic.groupby('mmsi')['lon'].shift().astype('float32')
df_dynamic['time_diff'] = df_dynamic.groupby('mmsi')['timestamp'].diff().astype('int64')
df_dynamic=df_dynamic.dropna()
df_dynamic=haversine_gpu(df_dynamic)
df_dynamic = df_dynamic.dropna() #Eliminar valores nulos
df_dynamic = calcular_trayectorias(df_dynamic)
df_paradas = df_dynamic[df_dynamic['is_stop']==1]
df_trayectorias = df_dynamic[df_dynamic['is_stop']==0]
parametros_trayectorias = calcular_parametros_trayectoria(df_trayectorias)
parametros_paradas = calcular_parametros_paradas(df_paradas)
parametros_cinematicos = calcular_parametros_cinematicos(parametros_trayectorias, df_dynamic, parametros_paradas)
parametros_cinematicos = parametros_cinematicos.fillna(0)
```

Figura 3-41 Implementación de la lectura y llamadas a funciones sin iteración por MMSI

4 RESULTADOS OBTENIDOS

En este capítulo se presentan y analizan en detalle los resultados obtenidos tras la aplicación de los desarrollos reseñados en el capítulo anterior del presente trabajo. El enfoque principal radica en la evaluación del rendimiento tanto predictivo como computacional de los algoritmos seleccionados, destacando especialmente la optimización lograda mediante el uso de procesamiento paralelo en GPU, con la biblioteca *cuDF*. Se ofrecen comparativas específicas entre diferentes métodos y técnicas empleadas, utilizando conjuntos de datos estáticos y dinámicos para una evaluación más completa.

Los resultados se dividen en dos categorías principales según la naturaleza de los datos utilizados: datos estáticos y datos dinámicos. Ambas categorías presentan problemáticas diferentes en cuanto a volumen, estructura y exigencias computacionales, permitiendo evaluar la robustez y escalabilidad de los algoritmos propuestos. Además, se discuten las ventajas obtenidas al migrar los procesos desde la CPU hacia la GPU, incluyendo una descripción cuantitativa de las mejoras en tiempo de ejecución y eficiencia general. Finalmente, el capítulo incluye un análisis comparativo que resume las diferencias observadas y ofrece una discusión en profundidad sobre la relevancia y el impacto de estos hallazgos dentro del contexto del análisis predictivo en aplicaciones marítimas.

Para la evaluación de los resultados se hará uso de las métricas definidas anteriormente (*accuracy*, *precision*, *recall* y *f1-score*).

4.1 Estructuración de los experimentos

En este TFG se ha perseguido el objetivo de mejorar la eficiencia computacional, pero se pueden distinguir dos vertientes: la primera y más importante, mejorar los tiempos de ejecución mediante el uso de otras bibliotecas y acceso a los datos de partida y la segunda, la mejora de la calidad de la predicción, mediante el uso de diferentes algoritmos de aprendizaje automático supervisado. Adicionalmente, se hace distinción entre los experimentos realizados con los datos estáticos y con los dinámicos.

En la Tabla 4-1 se compara el punto de partida obtenido de [1] y los resultados obtenidos en los diferentes experimentos. Cabe destacar que solo se expondrán aquellos experimentos en los que los resultados han sido positivos, estando el resto detallado en el capítulo previo.

Experimento	Algoritmo de aprendizaje automático	Cambio de parámetros	Resultados (accuracy)
Estudio previo de referencia [1]	<i>Random Forest</i>	-	86%
<i>Estáticos 0</i>	<i>Random Forest</i>	-	87%
<i>Estáticos 1</i>	<i>XGBoost</i>	Parámetros de partida	88%
<i>Estáticos 2</i>	<i>XGBoost</i>	<i>max_depth</i> = 10	88,36%
<i>Estáticos 3</i>	<i>XGBoost</i>	<i>eta</i> = 2	88,89%
<i>Estáticos 4</i>	<i>XGBoost</i>	Rondas = 650	89%

Tabla 4-1 Resumen de los experimentos realizados con los datos estáticos

Por otro lado, los experimentos realizados con los datos dinámicos están más enfocados a buscar la mejora de los tiempos de ejecución. Para ello, se realizan distintos experimentos probando bibliotecas y directivas e incluso distintas cantidades de parámetros calculados. Esta mejora de la eficiencia se ha realizado buscando mantener la calidad de la predicción. Para asegurar que no se pierde precisión se va mostrando, en cada caso, la calidad de la predicción obtenidas tras alimentar los modelos de predicción con los parámetros globales, es decir, la unión de los parámetros dinámicos con los estáticos.

Experimento	Técnica usada	Tiempo de extracción de parámetros dinámicos	Precisión <i>Random Forest</i>	Precisión <i>XGBoost</i>
Estudio previo de referencia [1]	Ejecución del código de [1] en el servidor antiguo	72 - 96 horas	90%	-
<i>Experimento datos globales 0: Comprobación de resultados en la nueva máquina</i>	Ejecución del código de [1] en el servidor actual	73 horas	90%	91%
<i>Aplicación de directiva</i>	<i>%load_ext cudf.pandas</i>	111 horas	-	-
<i>Primera iteración: Implementación del nuevo código con cuDF</i>	Bibliotecas <i>cuDF</i> compatibles con la GPU	16 horas	89,68%	91,39%
<i>Segunda iteración: Extracción de parámetros seleccionados</i>	Extracción de los parámetros con más importancia para cada algoritmo	2 horas (<i>Random Forest</i>)	89,73%	-
		5 horas (<i>Xgboost</i>)	-	91,11%
<i>Optimización del acceso a los datos</i>	Se elimina el bucle	10 segundos	90%	92%

Tabla 4-2 Resumen de los experimentos realizados con los datos dinámicos y la calidad de la predicción asociada a cada experimento con los parámetros globales

El objetivo de la Tabla 4-2 es poder consultar de manera visual y rápida las mejoras obtenidas en los resultados a lo largo de los diferentes experimentos llevados a cabo durante el desarrollo de este TFG. En ella, se detallan los experimentos y pruebas realizadas relacionadas con el procesamiento de los datos dinámicos. La primera fila representa el resultado obtenido en el TFG del curso 23/24 [1]. Esto sirve para poder comparar si los experimentos realizados consiguen mejora o no. En segundo lugar, se ejecutó el código de [1] obteniendo el resultado visible en la Tabla 4-2. A continuación, se aplica una directiva que no obtiene buenos resultados y, por tanto, se descarta y no se alimentan los algoritmos con los parámetros extraídos en este experimento. Es en este momento cuando se llega a la conclusión de que será necesario programar explícitamente el grueso del código principal y cambiar el enfoque. Para este experimento ya se mejoran los resultados francamente, pero a la vista del poco peso para la predicción de algunos parámetros, se decide extraer unos pocos parámetros en concreto. Es necesario adaptar el código para calcular los parámetros más relevantes en cada caso, aplicando el mismo proceso

tanto para el algoritmo *Random Forest* como para *XGBoost*, ajustando la selección de variables clave según la importancia que cada modelo les otorga.

Los parámetros dinámicos extraídos de los experimentos se utilizan para alimentar a ambos algoritmos implementados para la predicción junto con los estáticos para así comprobar y asegurar que las mejoras en los tiempos de ejecución no perjudican a los resultados obtenidos. La mejor de las opciones será siempre alimentar los algoritmos con los parámetros conjuntos (dinámicos y estáticos) y no únicamente con un grupo de ellos y, por esta razón, se representa en la Tabla 4-2 la precisión para los parámetros conjuntos y no dinámicos únicamente.

4.2 Análisis de los resultados

4.2.1 Mejora de la calidad de predicción en los datos estáticos con *XGBoost*

La ejecución del código empleado para obtener los parámetros estáticos fue extremadamente rápida, por lo que se decidió centrar los esfuerzos en esta categoría en mejorar la calidad de la predicción mediante el uso del algoritmo de aprendizaje automático supervisado *XGBoost*.

En el capítulo anterior se desarrollaron las iteraciones que se llevaron a cabo para optimizar *XGBoost*. Finalmente, tras varias iteraciones, la configuración óptima, con la que se obtiene el mejor resultado, emplea los siguientes valores de los parámetros usados:

- Porcentaje de datos usados para la prueba: 20%
- Profundidad de los árboles de decisión: 10
- Tasa de aprendizaje (*eta*): 0.2
- Número de rondas: 650

Aunque todo el código principal de procesamiento y extracción de parámetros se ha implementado utilizando la biblioteca *cuDF* para aprovechar la aceleración computacional que proporciona la GPU, la aplicación práctica de los modelos predictivos se ha realizado finalmente con la biblioteca *pandas*. Esto se debe a que, tras las pruebas realizadas, se observó que el modelo de predicción basado en *pandas* y ejecutado en la CPU proporcionó resultados un 1,4% más precisos en comparación con su equivalente implementado en *cuDF* sobre GPU. Esta diferencia es apreciable comparando los resultados obtenidos para *XGBoost* y para *Random Forest* en la Figura 4-1 y la Figura 4-2, respectivamente.

Precisión del modelo: 0.8864

Reporte de Clasificación:				
	precision	recall	f1-score	support
0.0	0.91	0.93	0.92	4002
1.0	0.87	0.86	0.87	988
2.0	0.77	0.77	0.77	527
3.0	0.88	0.83	0.85	1597
4.0	0.87	0.87	0.87	1057
accuracy			0.89	8171
macro avg	0.86	0.85	0.86	8171
weighted avg	0.89	0.89	0.89	8171

Figura 4-1 Resultados de la predicción en estáticos con *XGBoost* (*cuDF*)

Precisión del modelo: 0.8905

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.91	0.94	0.92	4003
1	0.87	0.89	0.88	988
2	0.81	0.79	0.80	527
3	0.89	0.81	0.85	1597
4	0.88	0.87	0.88	1057
accuracy			0.89	8172
macro avg	0.87	0.86	0.87	8172
weighted avg	0.89	0.89	0.89	8172

Figura 4-2 Resultados de la predicción en estáticos con XGBoost (pandas)

La diferencia observada en la precisión puede atribuirse principalmente a pequeñas variaciones numéricas introducidas por las operaciones en coma flotante que realizan las GPU, comparadas con aquellas efectuadas en CPU. Según estudios previos realizados [60], se ha demostrado que las GPU pueden presentar ligeras desviaciones en la precisión numérica debido a la naturaleza intrínseca del procesamiento paralelo. Estas operaciones en paralelo en GPU tienden a acumular y propagar errores de redondeo de manera distinta en comparación con las operaciones secuenciales típicas realizadas en CPU, especialmente relevantes cuando se manejan cálculos numéricos sensibles y precisos.

Adicionalmente, la biblioteca *pandas* presenta ventajas destacables desde un punto de vista técnico en términos de precisión numérica debido a su prolongado desarrollo, madurez y estabilidad en el manejo de datos tabulares. En [61] se argumenta que *pandas* ofrece un control más estricto sobre la manipulación, almacenamiento y gestión numérica de los datos, aspecto crucial en contextos críticos como la predicción, donde mínimas diferencias numéricas pueden implicar cambios significativos en los resultados.

Además, el ecosistema de *pandas* está respaldado por una comunidad robusta y una extensa documentación técnica, lo que garantiza que cualquier inconveniente relacionado con la precisión o compatibilidad numérica pueda abordarse de manera más eficiente y confiable. Por otro lado, aunque *cuDF* proporciona un notable incremento en velocidad de procesamiento y eficiencia computacional, especialmente en tareas relacionadas con la preparación y transformación de grandes volúmenes de datos, su madurez en términos de precisión numérica aún se encuentra ligeramente por detrás de *pandas*.

Por estos motivos técnicos y prácticos, la decisión final fue utilizar *cuDF* exclusivamente para la fase de preparación y procesamiento inicial de los datos, aprovechando la eficiencia en tiempos que ofrece la GPU. Posteriormente, para la fase final de predicción, se decidió utilizar *pandas*, priorizando así la precisión numérica y asegurando la obtención del resultado predictivo óptimo.

Comparando estos resultados con el punto inicial del estudio (86%) que fueron los resultados obtenidos en [1] y que se reflejan en la Figura 3-4, se aprecia una mejora sustancial en precisión hasta alcanzar un 89,05% en el *Experimento Estáticos 4*. Analizando detalladamente las métricas obtenidas (*precision*, *recall*, *f1-score*), se aprecia un incremento generalizado en los resultados del modelo con *XGBoost* respecto al modelo inicial basado en *Random Forest*. Esta mejora específica en cada categoría refleja la capacidad del algoritmo *XGBoost* para gestionar mejor datos desbalanceados y características correlacionadas, gracias a sus propiedades inherentes de optimización mediante gradiente y control de profundidad y tasa de aprendizaje.

En resumen, la mejora en precisión alcanzada por *XGBoost* sobre *Random Forest* en los datos estáticos no solo se debe a su naturaleza iterativa y su gestión eficiente del sobreajuste, sino también a su habilidad para manejar datos complejos y correlacionados de manera más eficiente.

4.2.2 Resultados del Experimento datos globales 0

Los resultados obtenidos ejecutando el código de [1] en una máquina con mayores capacidades no ha implicado ninguna mejora en los tiempos de ejecución. Mientras que en [1] se señalaba que los tiempos de ejecución se encontraban entre las 72 y 96 horas, con este experimento, nos hemos colocado en el límite inferior del rango, que aunque está cerca de los mejores tiempos de ejecución conseguidos el año pasado, no se obtiene una mejora como podría esperarse, necesitando un tiempo de ejecución de unas 73 horas aproximadamente tal y como se representó en el capítulo anterior en la Figura 3-11, estableciendo el punto de partida del presente TFG.

Al no obtener mejoras con respecto a la ejecución en el equipo del año pasado se exploran distintas líneas de acción para conseguir explotar las capacidades computacionales de las que disponemos.

4.2.3 Aplicación de la directiva `%load_ext cudf.pandas`

El primer experimento realizado es la aplicación de la directiva `%load_ext cudf.pandas` cuyo funcionamiento se explicó previamente en el apartado 3.3.3.2.

El resultado del tiempo de ejecución del bucle con esta directiva se reseña en la Figura 4-3. Como se puede observar, se han visto incluso aumentados los tiempos de ejecución con respecto al punto de partida que no usaba esta directiva.

```
Processing MMSIs: 100%|██████████| 40859/40859 [111:26:27<00:00, 9.82s/it]
Existen 94 MMSI cuyo df contiene Nan.
Existen 1787 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 401004.8244175911 segundos.
```

Figura 4-3 Tiempos de ejecución con la directiva `%load_ext cudf.pandas`

Como se ha expuesto anteriormente, esta directiva trata de usar funciones compatibles con *cuDF* que sean similares a las usadas en el código mediante *pandas*. En caso de que falle en la ejecución en la GPU o que no existan funciones similares a las de *pandas* en *cuDF*, se hace una copia temporal en CPU y se aplica una segunda ejecución con *pandas*, para su posterior copia en la GPU con *cuDF*. Esto da lugar a una doble ejecución para la misma línea de código y a transformaciones continuas de datos de *cuDF* a *pandas* y viceversa, lo que provoca una pérdida en la eficiencia computacional y alargando los tiempos de ejecución.

El código en el que se ha aplicado la optimización fue desarrollado utilizando funciones y bibliotecas propias de *pandas* o compatibles con él. Los prolongados tiempos de ejecución, que alcanzan hasta 111 horas, sugieren que la mayoría de las funciones empleadas no son compatibles con *cuDF*. Como consecuencia, la paralelización en la GPU, objetivo principal de la migración, apenas se lleva a cabo en el código, lo que limita significativamente las mejoras en el rendimiento.

Si bien *cuDF* es una herramienta potente para acelerar el procesamiento de datos mediante el uso de la GPU, su principal limitación radica en la falta de compatibilidad con muchas funciones que sí están disponibles en *pandas*. Un factor determinante en el aumento del tiempo de ejecución podría ser el cálculo y discretización de las trayectorias, ya que este proceso, además de ser computacionalmente exigente, se basa en funciones de *geopandas* y *movingpandas*, las cuales no son compatibles con *cuDF* ni cuentan con alternativas directas en esta biblioteca. Esto implica que la mayor parte del procesamiento se sigue realizando en la CPU, anulando los beneficios inicialmente esperados de la migración a GPU.

4.2.4 Primera iteración

En esta primera iteración se concluyó que sería necesario la implementación de funciones con el objetivo de extraer los parámetros mediante el uso de la GPU con bibliotecas de *cuDF* que mejoraran la

eficiencia computacional del código. Se ha tenido presente constantemente que, aunque el objetivo principal fuera mejorar la eficiencia computacional para reducir los tiempos de ejecución, no se podían admitir pérdidas relevantes en la calidad de predicción.

Durante la primera iteración, se desarrolla un código con el que se extraen aquellos parámetros obtenidos en el Experimento Dinámicos 4 del código desarrollado previamente en [1]. De esta manera es posible comparar directamente las mejoras en rendimiento computacional obtenidas respecto al Experimento Dinámicos 4 de [1]. La motivación principal para realizar este experimento era lograr una reducción significativa en los tiempos de procesamiento, aprovechando la capacidad de procesamiento paralelo de la GPU mediante la biblioteca *cuDF*, sin comprometer la calidad de las predicciones siguiendo con el formato de bucle que se venía utilizando. El uso del bucle se debe a que inicialmente se disponía de los datos de los mensajes separados por MMSI, es decir, por buque en diferentes archivos, haciendo necesario el uso de un bucle para leer el contenido de todos los archivos.

En esta iteración se implementaron todas las funciones necesarias para la extracción de parámetros con *cuDF*. Los resultados mostraron un notable incremento en la eficiencia computacional, disminuyendo significativamente el tiempo de ejecución desde 73 horas en la iteración de referencia hasta cerca de 16 horas en esta primera iteración con *cuDF*, tal y como se observa en la Figura 4-4. Esta reducción tan sustancial en los tiempos de procesamiento se logró gracias a la capacidad de la GPU para realizar múltiples operaciones en paralelo sobre grandes conjuntos de datos.

```
Processing MMSIs: 100%|██████████| 40859/40859 [16:33:22<00:00, 1.46s/it]
Existen 0 MMSI cuyo df contiene NaN.
Existen 3719 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 59381.09842133522 segundos.
```

Figura 4-4 Tiempos de ejecución de la primera iteración con *cuDF*

En cuanto a la calidad de predicción, esta primera iteración logró mantener una precisión del 90%, idéntica a la obtenida en [1]. Esto confirmó que la migración a *cuDF* no afectó negativamente la calidad predictiva y que, por tanto, era posible combinar beneficios significativos en eficiencia computacional con una precisión predictiva elevada.

Esta iteración estableció claramente la ventaja del uso de *cuDF* en escenarios con grandes volúmenes de datos dinámicos, validando su utilidad en aplicaciones donde los tiempos de respuesta y el rendimiento computacional son críticos.

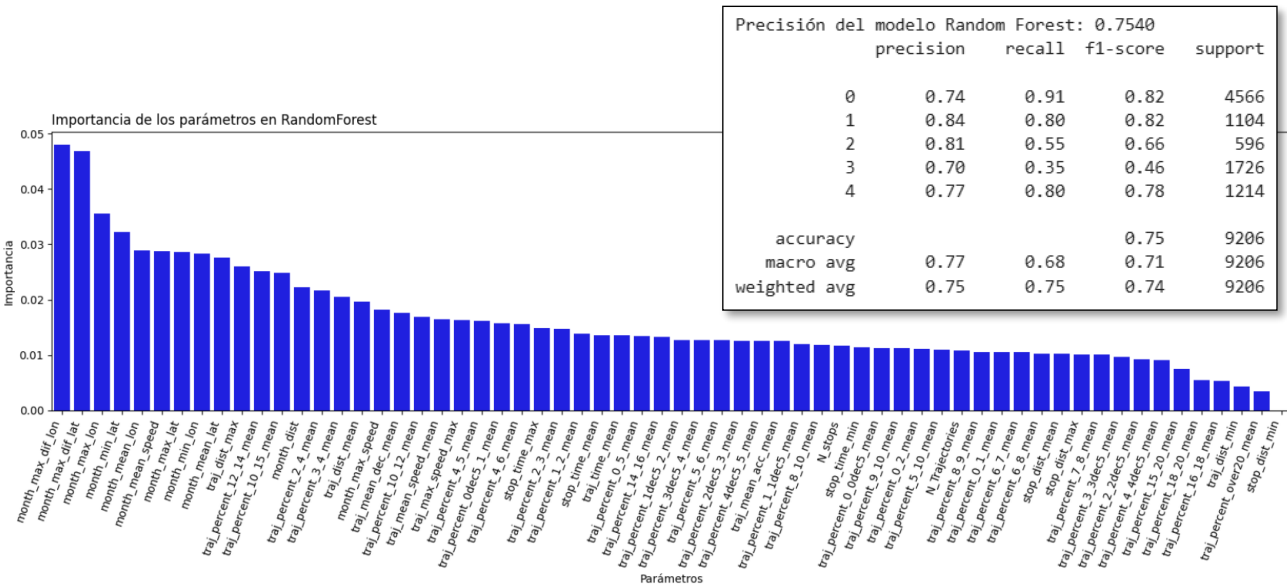


Figura 4-5 Resultados del Experimento datos globales 1 con Random Forest

Para evaluar la calidad predictiva de esta primera iteración se utilizó *Random Forest* para poder comparar directamente con las precisiones obtenidas en [1]. El *Experimento datos globales 1* consiste en la predicción del campo *shiptype* usando únicamente los parámetros cinemáticos.

Como se puede observar en la Figura 4-5, la precisión promedio fue del 76%, con valores de *precision* y *recall* variables según las diferentes categorías analizadas. La precisión más alta fue obtenida para la categoría "1" (como recordatorio, en la Tabla 3-3 se define el mapeo realizado en el campo *shiptype*, la categoría 1 corresponde a un *ShipType.Fishing*) con un valor del 84%, mientras que el *recall* más alto se obtuvo en la categoría 0: *ShipType.Cargo* (91%). La gráfica de importancia relativa de parámetros indica que las variables más influyentes en la predicción fueron aquellas relacionadas con la variación máxima de latitud y longitud mensual (*month_max_dif_lat* y *month_max_dif_lon*), así como la velocidad media mensual (*month_mean_speed*).

Estas métricas revelan que *Random Forest*, aunque proporciona resultados razonablemente buenos, no alcanza la precisión obtenida por *XGBoost*. Gracias a la experiencia obtenida en los experimentos estáticos, se decidió obtener la predicción del campo *shiptype* mediante este algoritmo. Una vez más, se han obtenido mejores resultados, reafirmando nuevamente la elección preferente de *XGBoost* para la fase final del modelo predictivo debido a su robustez y mejor gestión de variables correlacionadas y datos complejos. En la Figura 4-6 se detallan los resultados obtenidos con *XGBoost*.

La precisión promedio alcanzada con *XGBoost* fue del 80%, superando en un 4% la obtenida con *Random Forest*. Las métricas específicas por categoría indican una mejora significativa en *recall* y *f1-score*, especialmente destacable en las categorías *ShipType.Cargo* (0) y *ShipType.Fishing* (1). Al observar la importancia relativa de las características según *XGBoost*, se aprecia un cambio sustancial respecto a *Random Forest*. La principal razón para estos cambios radica en la metodología interna de cada algoritmo para evaluar y seleccionar características: *XGBoost* emplea una estrategia basada en la optimización mediante gradientes y ganancia por división (*importance_type='gain'*), mientras que *Random Forest* utiliza métricas basadas en impureza, como el índice de *Gini*. Estas diferencias metodológicas hacen que *XGBoost* sea más sensible a parámetros directamente relacionados con la dinámica y velocidad de los buques, priorizando parámetros cinemáticos como la diferencia máxima de latitud, velocidad promedio mensual y características relacionadas con trayectorias específicas.

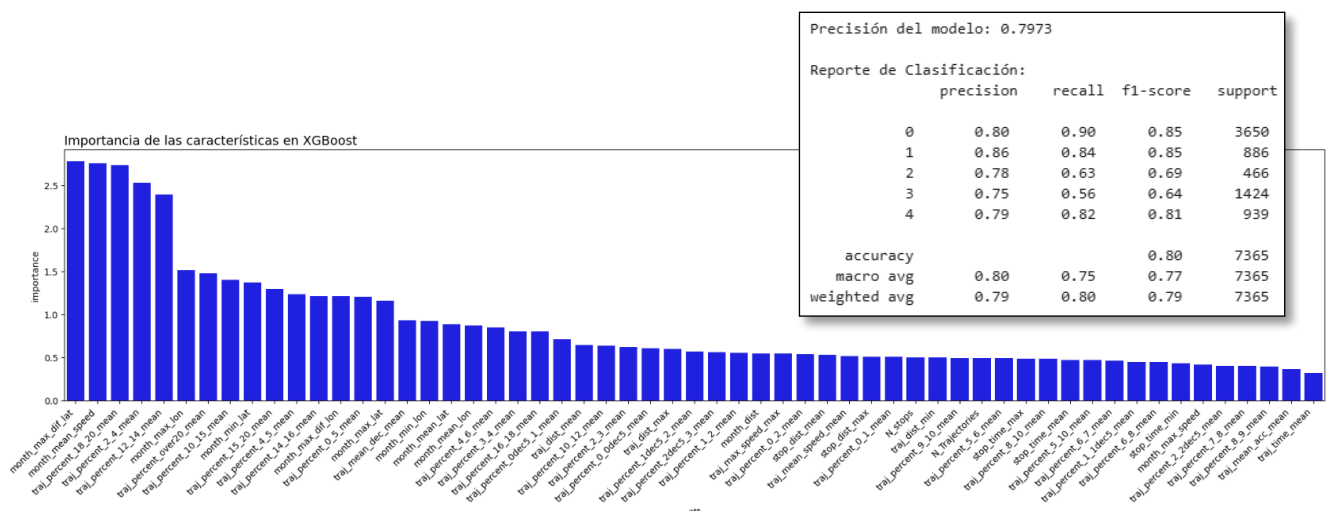


Figura 4-6 Resultados del Experimento datos globales 1 con XGBoost

En *Random Forest*, la importancia de una variable se determina a partir del *índice de Gini*, el cual mide la impureza de los datos en cada nodo del árbol de decisión. La impureza de Gini indica cuán mezcladas están las clases dentro de un nodo; si un nodo contiene datos de una sola clase, su impureza es cero, mientras que, si tiene datos de varias clases, su impureza es mayor. Cada vez que una variable se usa para dividir un nodo en un árbol de decisión dentro de *Random Forest*, contribuye a la reducción

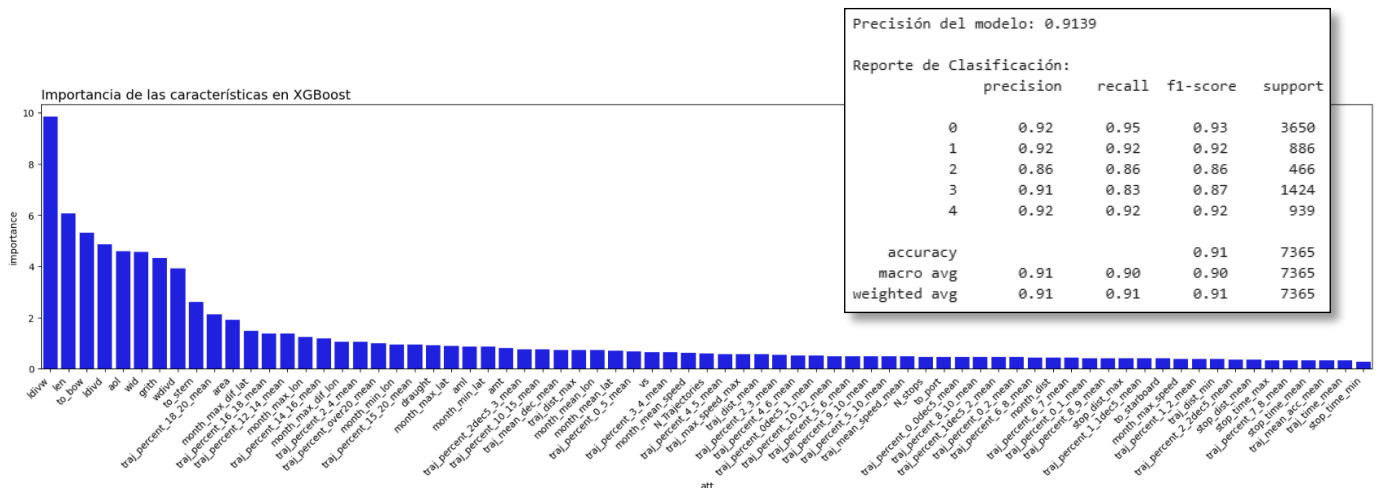


Figura 4-8 Resultados del Experimento datos globales 2 con XGBoost

4.2.5 Segunda iteración

En esta segunda iteración se pretende reducir los tiempos mediante la selección óptima del número de parámetros extraídos. Se eliminan las columnas con menos importancia para ambos algoritmos tal y como se desarrolla en el apartado 3.3.3.5 modificando las funciones para que calculen única y exclusivamente los parámetros deseados.

En la primera reducción de parámetros (29 parámetros eliminados) se redujo el tiempo de ejecución de más de 16 horas a 7 horas y 57 minutos (ver Figura 4-9). En una segunda reducción de parámetros, se extraen los 20 parámetros de mayor relevancia en la predicción, a la vez que se mantiene la precisión de la predicción consiguiendo unos tiempos de ejecución de 2 horas y 29 minutos (ver Figura 4-10).

```
Processing MMSIs: 100%|██████████| 40859/40859 [7:57:25<00:00, 1.43it/s]
Existen 0 MMSI cuyo df contiene NaN.
Existen 3719 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 28409.429589509964 segundos.
```

Figura 4-9 Tiempo de ejecución de la segunda iteración, 29 parámetros eliminados para Random Forest

```
Processing MMSIs: 100%|██████████| 40859/40859 [2:29:23<00:00, 4.56it/s]
Existen 0 MMSI cuyo df contiene NaN.
Existen 3719 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 8727.39570403099 segundos.
```

Figura 4-10 Tiempo de ejecución de la segunda iteración, 42 parámetros eliminados para Random Forest

Comparando las gráficas que resumen la importancia de los parámetros para Random Forest y XGBoost (ver la Figura 4-7 y la Figura 4-8 respectivamente) se aprecia una ligera variación en el orden de importancia. Es por ello que para cada algoritmo se descartan distintas columnas. A la hora de alimentar el algoritmo de XGBoost fue necesario modificar las funciones para obtener los parámetros que para XGBoost eran más importantes. Una vez modificadas las funciones, se vuelve a ejecutar el bucle obteniendo un tiempo de ejecución de 5 horas, como se aprecia en la Figura 4-11. Aunque se calculen el mismo número de parámetros, estos 20 parámetros tardan más en extraerse, esto se debe a que los parámetros deseados para Random Forest se calculan de forma más eficaz por las operaciones necesarias para su cálculo que se realizan en el código principal.

```
Processing MMSIs: 100%|██████████| 40859/40859 [5:18:05<00:00, 2.14it/s]
Existen 0 MMSI cuyo df contiene NaN.
Existen 3719 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 18844.739042520523 segundos.
```

Figura 4-11 Tiempos de ejecución tras la reducción de parámetros para XGBoost

En esta iteración se han llevado a cabo dos experimentos como en la anterior: en el *Experimento datos globales 3*, el modelo se alimenta con los 20 parámetros cinemáticos con más importancia y en el *Experimento datos globales 4* que se alimenta al modelo con los parámetros dinámicos junto con los estáticos.

Al reducir el número de parámetros dinámicos significativamente, la precisión del modelo siendo alimentado solo por estos se ve reducida. Aunque esta reducción es notable, lo que realmente nos importa es la precisión al usar todos los parámetros, por tanto, se esperará a valorar el resultado de esta predicción una vez obtenidos los resultados para la predicción con los parámetros cinemáticos junto a los estáticos.

En la Figura 4-12 se observan los resultados obtenidos del *Experimento datos globales 3* junto con la gráfica en la que se representa la importancia de cada parámetro mediante el uso de *Random Forest*. La precisión (*Accuracy*) obtenida es del 73%, comparándola con la obtenida en el *Experimento datos globales 1* (ver Figura 4-5) que es de un 76%, se ve que las pérdidas son considerables. No obstante, como se estableció en el capítulo anterior, las pérdidas en la calidad de la predicción dejarán de ser asumibles cuando la precisión baje un 1% al usar para la predicción el conjunto de parámetros (dinámicos y estáticos), por tanto, se esperará al siguiente experimento que utiliza este conjunto de parámetros para establecer si los resultados son válidos o las pérdidas exceden el límite impuesto.

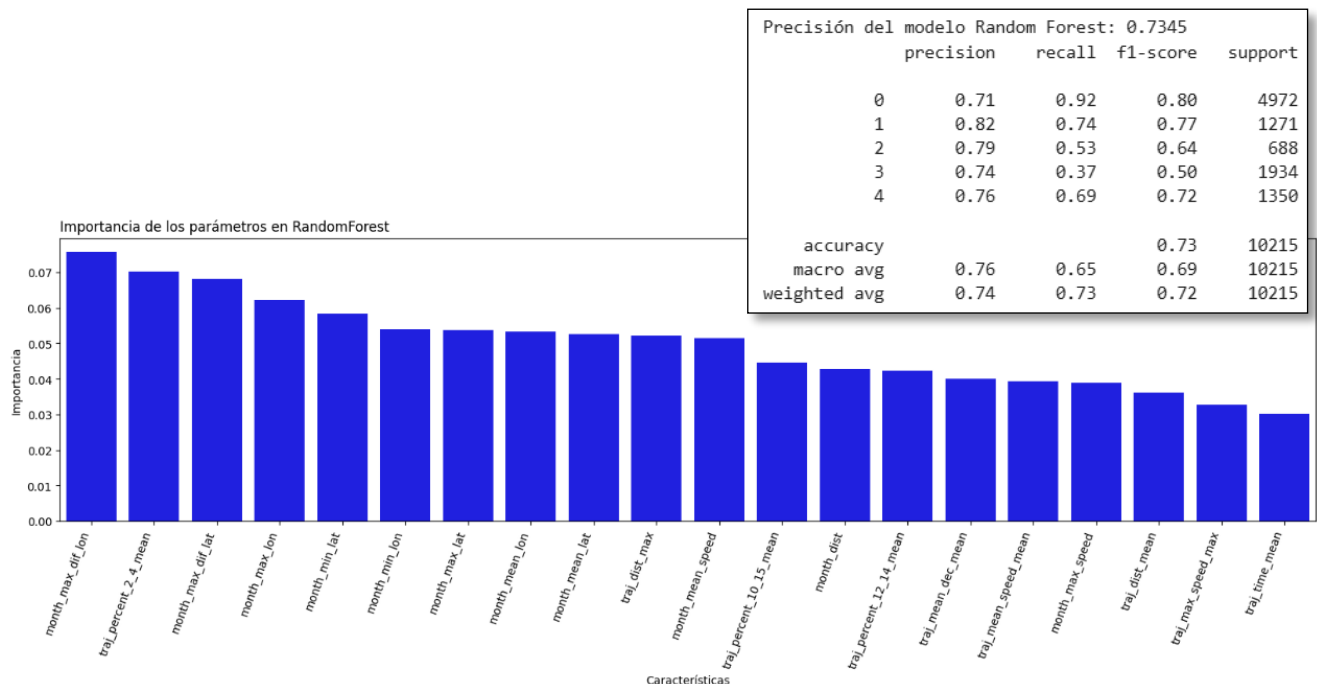


Figura 4-12 Resultados del Experimento datos globales 3 con Random Forest

Al utilizar XGBoost en el *Experimento datos globales 3*, ocurre algo similar, ya que también se observan pérdidas en la precisión. En la Figura 4-13 se reflejan los resultados del *Experimento datos globales 3* con XGBoost y la gráfica de importancia de los parámetros. Comparándolos con los resultados del *Experimento datos globales 1* con XGBoost (ver Figura 4-6), se observa que vuelve a haber una pérdida como en el caso de *Random Forest*. Esta vez las pérdidas son del 5% pero, como se dijo previamente, en ningún caso nos interesa predecir el campo *shiptype* únicamente con los parámetros

cinemáticos, sino que se usarán junto con los estáticos para la predicción y será entonces cuando se establezca si las pérdidas son asumibles o no.

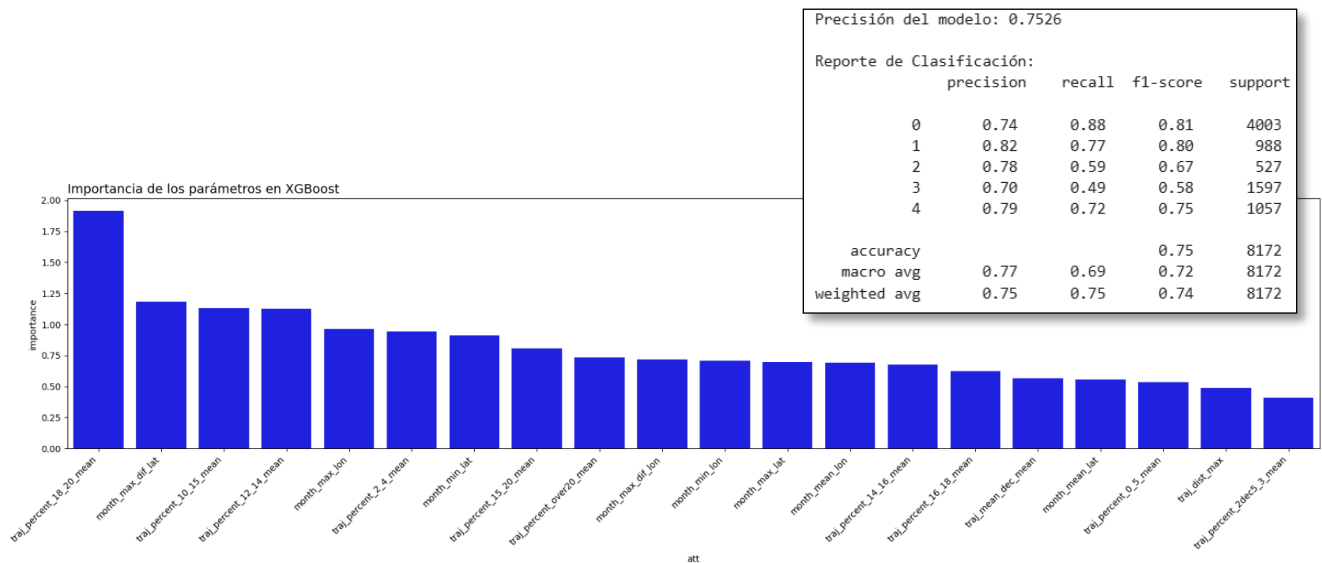


Figura 4-13 Resultados del Experimento datos globales 3 con XGBoost

Comparando los resultados obtenidos en el *Experimento datos globales 3* mediante los distintos algoritmos, se observa que *XGBoost* supera a *Random Forest* ofreciendo hasta un 2% más de precisión cuando se alimenta el modelo solo con los parámetros cinemáticos.

Siguiendo con la segunda iteración, se exponen los resultados del *Experimento datos globales 4*. En este experimento se alimentan ambos algoritmos con los parámetros dinámicos junto a los estáticos. Como en un entorno real la predicción siempre se realizará con el conjunto completo de parámetros, es en este experimento donde se debe valorar si las pérdidas son asumibles o no.

En la Figura 4-14 y en la Figura 4-15 se observan los resultados obtenidos en el *Experimento datos globales 4* con *Random Forest* y con *XGBoost*, respectivamente. Comparando los resultados obtenidos con ambos modelos con sus respectivos *Experimento datos globales 2* (ver Figura 4-7 y Figura 4-8) se concluye que la alteración de la precisión está dentro del límite establecido y, por ende, los resultados se dan como válidos. A mayores, comparando los resultados de este experimento con cada modelo, la precisión obtenida con *XGBoost* supera en más de un 1% respecto a los resultados obtenidos con *Random Forest*.

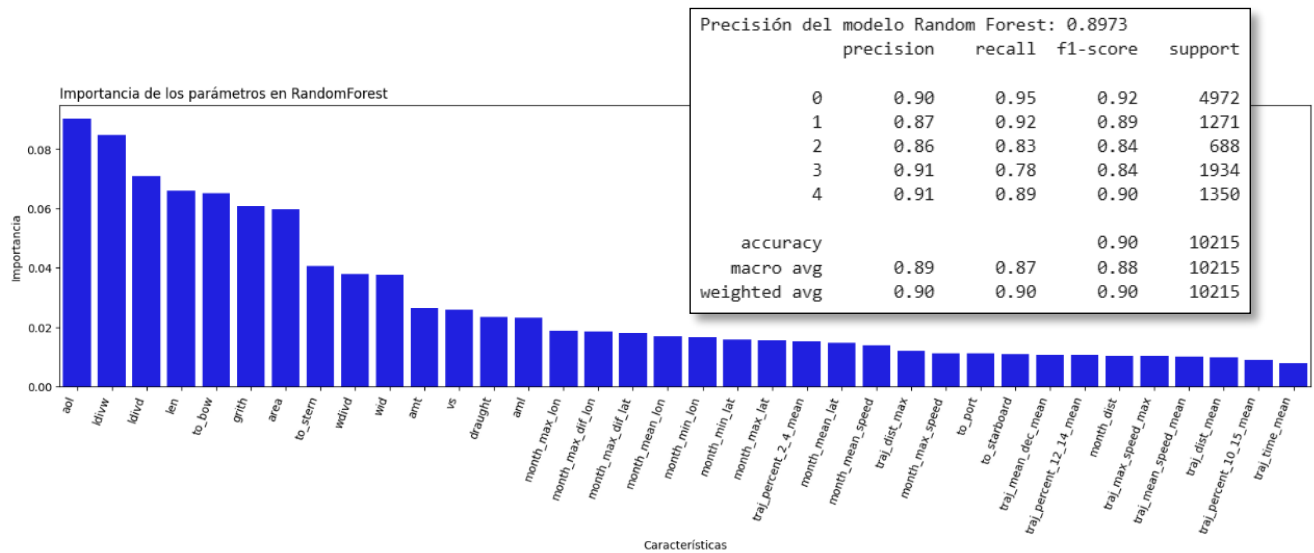


Figura 4-14 Resultados del Experimento datos globales 4 con Random Forest

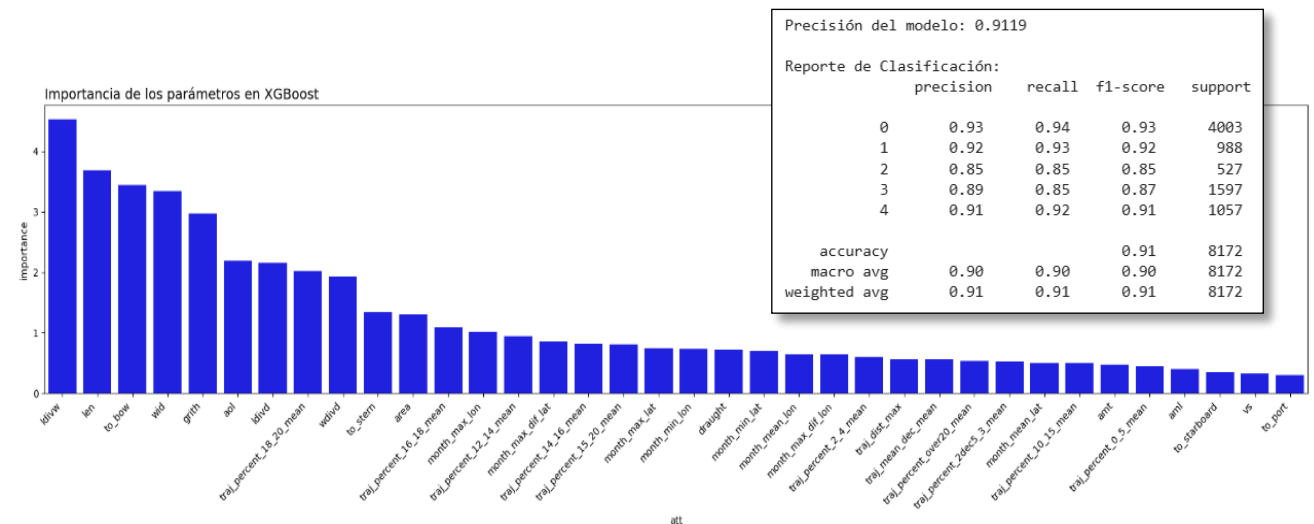


Figura 4-15 Resultados del Experimento datos globales 4 con XGBoost

4.2.6 Optimización del acceso a los datos

En este último experimento, se ha cambiado el enfoque sobre el acceso a los datos. En este apartado, se verán los resultados obtenidos tras aplicar este cambio. Para la medición de los tiempos se utilizará, en este caso, la directiva de *Jupyter %%time*, que muestra el tiempo de ejecución del código de la celda.

En la Figura 4-16 se muestra el tiempo que ha tardado en ejecutarse el procesamiento de los datos y la obtención de *parametros_cinematicos* en el que se recogen los mismos parámetros por buque que se calcularon en la primera iteración. Los resultados muestran que la reducción de los tiempos de ejecución ha sido notable tal y como se justifica en la Figura 4-16 con un tiempo de ejecución (*Wall time*) de 10.6 segundos.

```
%%time
df_dynamic = cudf.read_csv('dynamic.csv')
df_dynamic.reset_index(inplace=True)
df_dynamic=df_dynamic.drop('Unnamed: 0',axis=1)
df_dynamic = clean(df_dynamic)
df_dynamic['lat_shift'] = df_dynamic.groupby('mmsi')['lat'].shift().astype('float32')
df_dynamic['lon_shift'] = df_dynamic.groupby('mmsi')['lon'].shift().astype('float32')
df_dynamic['time_diff'] = df_dynamic.groupby('mmsi')['timestamp'].diff().astype('int64')
df_dynamic=df_dynamic.dropna()
df_dynamic=haversine_gpu(df_dynamic)
df_dynamic = df_dynamic.dropna() #Eliminar valores nulos
df_dynamic = calcular_trayectorias(df_dynamic)
df_paradas = df_dynamic[df_dynamic['is_stop']==1]
df_trayectorias = df_dynamic[df_dynamic['is_stop']==0]
parametros_trayectorias = calcular_parametros_trayectoria(df_trayectorias)
parametros_paradas = calcular_parametros_paradas(df_paradas)
parametros_cinematicos = calcular_parametros_cinematicos(parametros_trayectorias, df_dynamic, parametros_paradas)
parametros_cinematicos = parametros_cinematicos.fillna(0)

CPU times: user 11.3 s, sys: 4.68 s, total: 16 s
Wall time: 10.6 s
```

Figura 4-16 Tiempo de ejecución del código de procesamiento de datos en *df_dynamic*

La carga inicial del DF se ha convertido en un único proceso de lectura, permitiendo que las operaciones posteriores se realicen directamente sobre la memoria sin necesidad de acceder repetidamente al disco. Esto ha resultado en la reducción de tiempos de ejecución a segundos lo que permite el procesamiento y manejo de datos en tiempo real. En la Tabla 4-3 se comparan los tiempos de ejecución en tres escenarios: el punto de partida con procesamiento en *pandas* y acceso iterativo a los datos, la reprogramación en *cuDF*, pero manteniendo la lectura y proceso iterativo por MMSI, y finalmente, la optimización del acceso a los datos mediante la carga única en un DF global. Los resultados muestran una reducción drástica en los tiempos de ejecución, ya que operar sobre un único DF ha permitido aprovechar al máximo la computación en GPU, eliminando los cuellos de botella causados por la iteración sobre archivos individuales.

Método	Tiempo de ejecución
Acceso iterativo a archivos en CPU (<i>pandas</i>)	73 horas 22 minutos 7 segundos
Acceso iterativo a archivos en GPU (<i>cuDF</i>)	16 horas 33 minutos 22 segundos
Carga unificada de los datos en un único DF	10.6 segundos

Tabla 4-3 Comparación de los tiempos de ejecución en el procesamiento de los datos dinámicos

Este cambio no solo transforma por completo la eficiencia del sistema, sino que redefine la forma en la que se pueden ejecutar experimentos y realizar ajustes en el modelo, permitiendo una iteración prácticamente inmediata y haciendo viable la aplicación de este enfoque en entornos reales en tiempo real ya que el tiempo de ejecución es de unos 10 segundos.

Esta solución no es escalable, ya que si los datos leídos del fichero fueran mucho mayores (por ejemplo, transmisiones más frecuentes) o de un rango temporal mayor no se podría hacer porque, a pesar de la mejora en el acceso a los datos, esta estrategia ha generado un aumento en el consumo de la memoria dificultando la tarea y obligando a optimizar la gestión de los recursos de memoria. Esta

limitación justifica la necesidad de los experimentos anteriores en los que se mantiene el proceso iterativo, que, aunque requiere de mayores tiempos de ejecución, es escalable. No obstante, los problemas de memoria fueron minimizados consiguiendo que se pudiera ejecutar el código sin problemas y, a mayores, se consiguió guardar los resultados de *parametros_cinematicos* en un archivo CSV.

A continuación, se alimentan ambos algoritmos (*Random Forest* y *XGBoost*) con el DF obtenido de este experimento tal y como se ha procedido en el resto de los experimentos con el fin de comprobar que la calidad de la predicción no empeora.

En el *Experimento datos globales 5* se obtienen los resultados para ambos algoritmos con los parámetros dinámicos obtenidos con la carga unificada de los datos en un único DF. En la Figura 4-17 se observan los resultados obtenidos para este experimento con *Random Forest*, donde se mejora la precisión del modelo (que hace referencia a *accuracy* de las métricas, pero aportando más decimales) en casi un 0,5% la precisión obtenida en el *Experimento datos globales 1* con este mismo algoritmo (ver Figura 4-5). Mientras que en la Figura 4-18 se representan los resultados obtenidos con *XGBoost* para el mismo experimento, obteniendo una mayor mejora, mejora que supone aumentar en un con respecto a su igual en el *Experimento datos globales 1* (ver Figura 4-6). El motivo por el que la predicción mejora en este experimento con ambos modelos es por la mayor precisión a la hora de detectar las paradas. La mejora en *XGBoost* se cuantifica en un 1% en este experimento frente al *Experimento datos globales 1*.

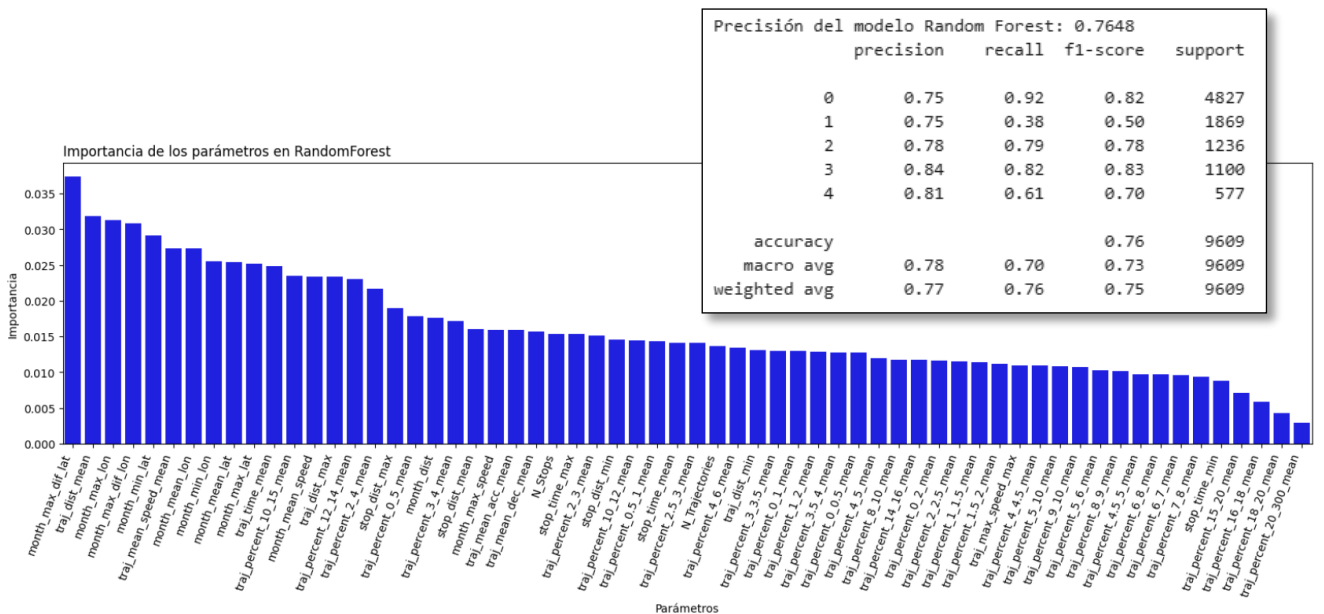


Figura 4-17 Resultados del Experimento datos globales 5 con Random Forest

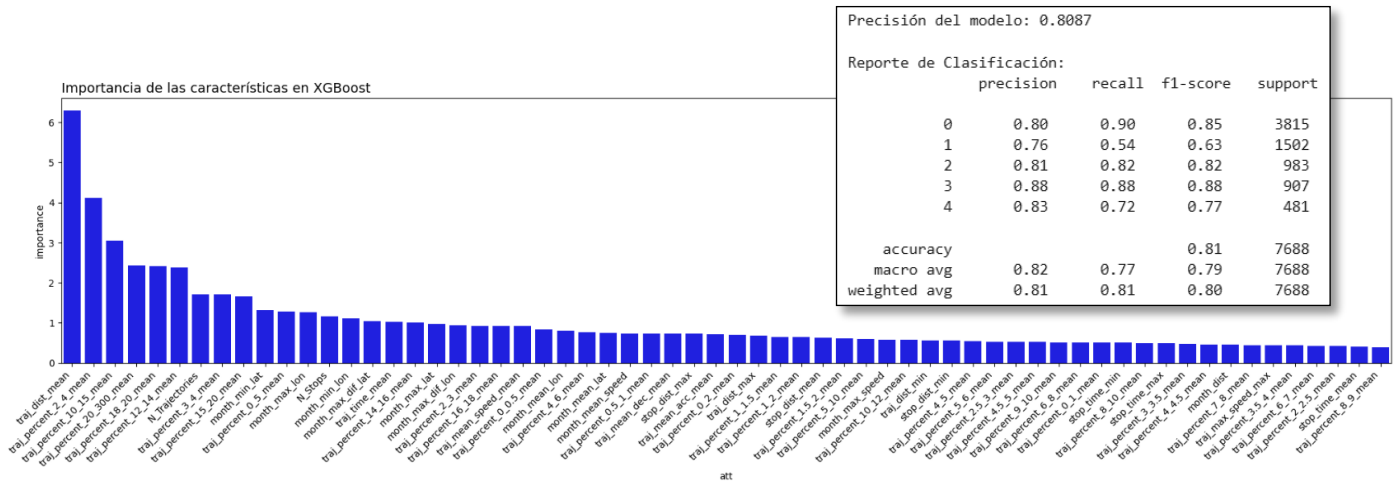


Figura 4-18 Resultados del Experimento datos globales 5 con XGBoost

Finalmente, el *Experimento datos globales 6* es el último experimento realizado. En este experimento, se alimentan los algoritmos con los mismos datos dinámicos que en el experimento anterior, pero se añaden los estáticos. Este experimento es de vital importancia debido a que la precisión obtenida en este marcará si este experimento es válido o la calidad de la predicción empeora y no es asumible. En la Figura 4-19 y la Figura 4-20 se representan los resultados para este experimento con *Random Forest* y *XGBoost* respectivamente.

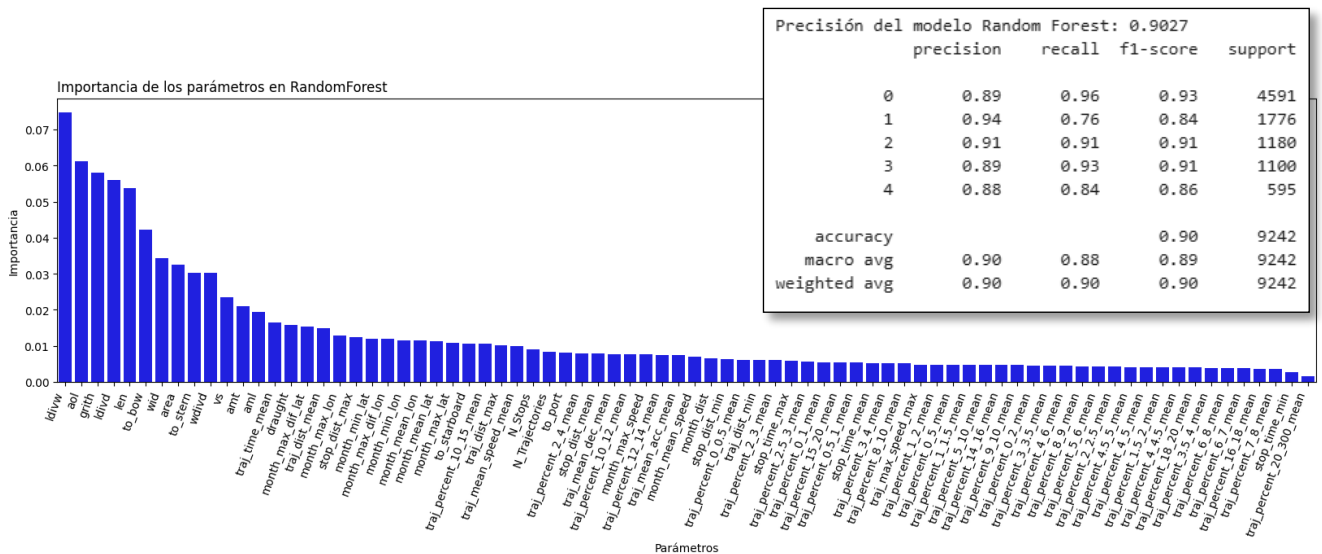


Figura 4-19 Resultados del Experimento datos globales 6 con Random Forest

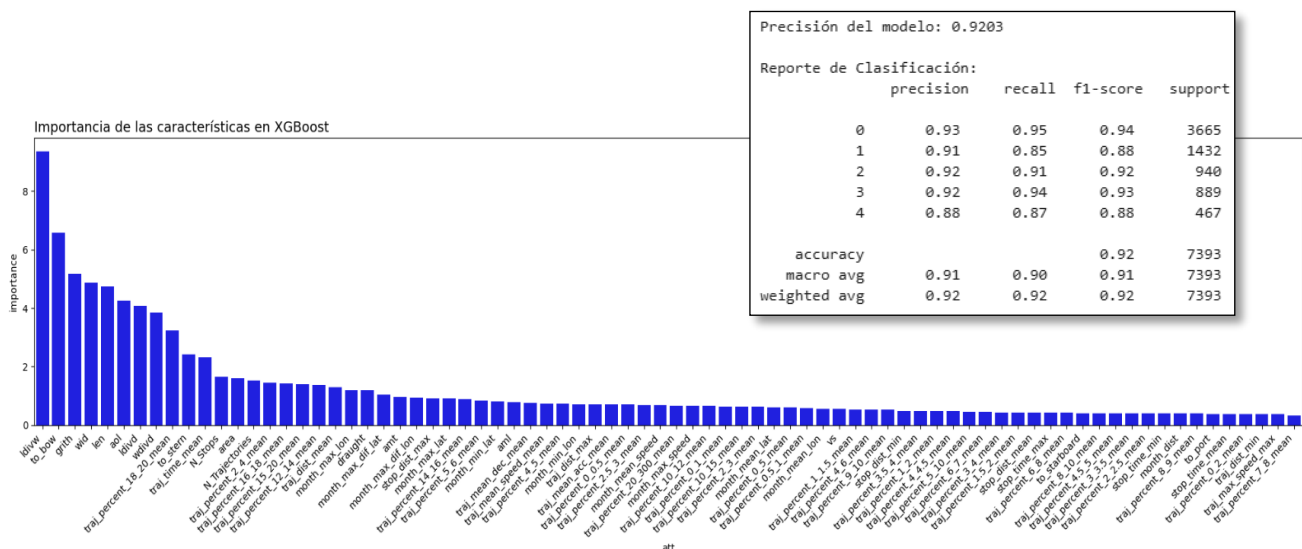


Figura 4-20 Resultados del Experimento datos globales 6 con XGBoost

Finalmente, los resultados obtenidos en el *Experimento datos globales 6* con *XGBoost* es el más favorable. En este experimento se ha conseguido aumentar la precisión desde un 90% que se obtuvo en [1] hasta un 92% gracias a la implementación del algoritmo *XGBoost* y la mejora en la detección de paradas. Por otro lado, se han conseguido reducir los tiempos de ejecución en la extracción de parámetros desde las 73 horas que se establecieron como punto de partida hasta los 10 segundos aproximadamente en calcular todos los parámetros en un DF único.

4.2.7 Justificación de la importancia de las columnas en la predicción

Con el objetivo de comprender los resultados y la importancia que los modelos de predicción le dan a cada parámetro, se analizan cómo varían sus distribuciones según el tipo de buque. En este apartado, se emplean gráficas de densidad que permiten visualizar la distribución de un parámetro específico para cada tipo de buque, facilitando la identificación de patrones diferenciadores que aporten al modelo la capacidad de distinguir entre un tipo de buque y otro.

El objetivo de este análisis es determinar qué variables presentan diferencias significativas entre clases y, por lo tanto, aportan información valiosa para la clasificación. Si una variable tiene distribuciones muy similares para todos los tipos de buques, su capacidad discriminativa será baja, mientras que, si muestra diferencias claras entre clases, será un indicador relevante para el modelo de predicción.

Estas visualizaciones permiten evaluar el grado de separabilidad de los datos, identificando qué variables pueden aportar más información a la predicción del tipo de buque. Uno de los aspectos clave en este análisis es la separabilidad de las distribuciones. Si las curvas de densidad de los diferentes tipos de buques presentan diferencias claras y están bien diferenciadas entre sí, significa que el parámetro tiene un alto poder discriminativo y puede contribuir significativamente al modelo de clasificación. Por el contrario, cuando se observa un alto grado de solapamiento entre clases, esto indica que la variable no es suficientemente informativa para distinguir entre ciertos tipos de buques y, por lo tanto, su inclusión en el modelo podría no aportar mejoras sustanciales en la predicción. Además, se analizarán la tendencia central y la dispersión de cada variable, evaluando si las medias y varianzas difieren significativamente entre clases. Si estas diferencias son notables, el parámetro en cuestión puede proporcionar patrones útiles para mejorar la precisión del modelo.

En la Figura 4-21 se muestra este análisis aplicado al parámetro *ldivw*. Recordamos que este parámetro representa el cociente entre la eslora y la manga. Se observa que algunas categorías de buques (*Cargo*, *Fishing* y *Passenger*) presentan distribuciones bien definidas y diferenciadas, lo que hace que este parámetro sea valioso para la predicción, de hecho, es el parámetro con mayor relevancia para el

Experimento datos globales 6 con XGBoost (ver Figura 4-20). Aunque los *shiptype Tug* y *Tanker* tengan una distribución menos caracterizada, este parámetro le sirve al modelo para discretizar los buques que sí tienen una distribución característica.

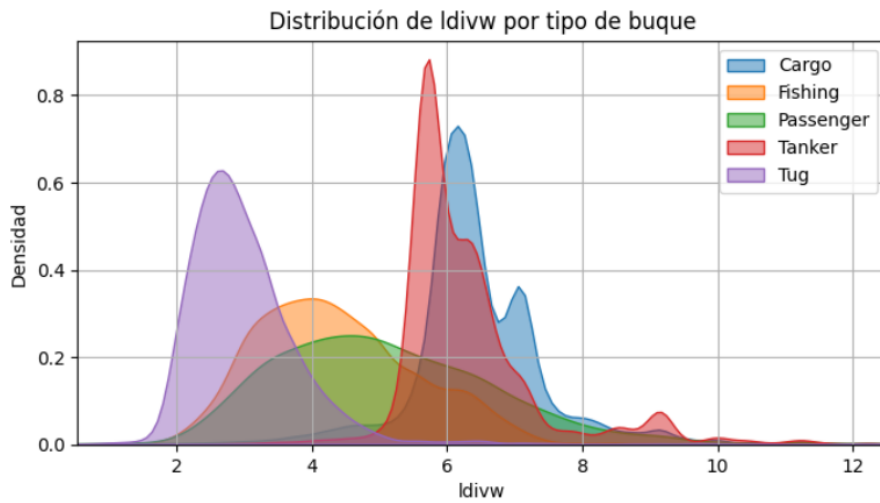


Figura 4-21 Gráfica de densidad del parámetro *ldivw*

A continuación, se recogen algunas de las gráficas de densidad de los parámetros más determinantes en el último experimento realizado (ver Figura 4-22). Se observa cómo los parámetros estáticos son más relevantes que los dinámicos, siendo estos últimos más difícil de diferenciar. En todas ellas, se observa cómo hay al menos un buque que destaca sobre el resto. El modelo no necesita parámetros que discreticen a la perfección entre todos los tipos de buques gracias al gran número de parámetros calculados, sino que con diferenciar uno del resto ya está aportando mucha información para la clasificación.

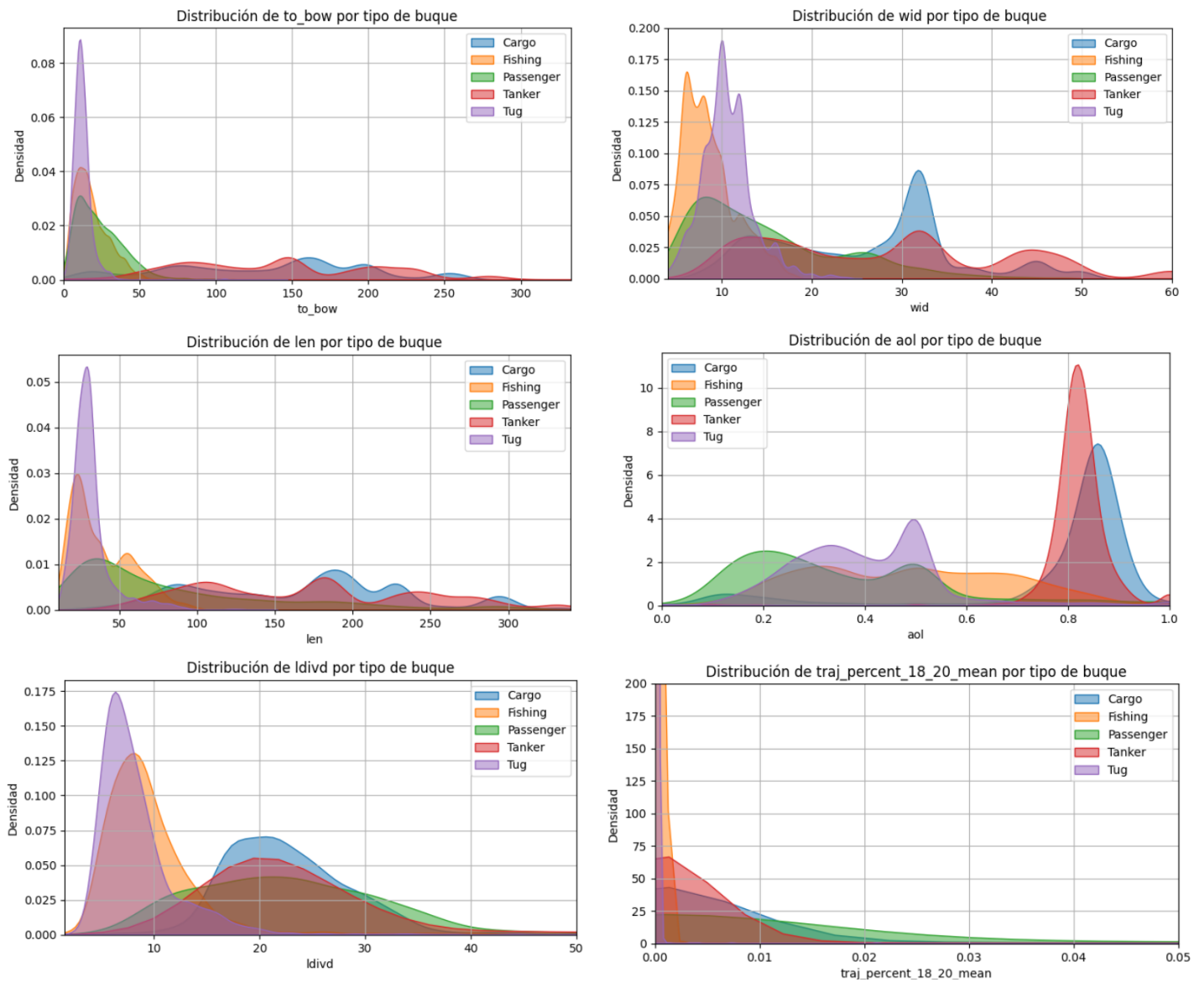


Figura 4-22 Gráficas de densidad para distintos parámetros

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

El objetivo principal del presente TFG era la mejora de la eficiencia computacional de un sistema de predicción del tipo de buque. Dicho objetivo se refiere a la optimización del uso de los recursos de hardware y software con el fin de reducir tiempos de ejecución, consumo de memoria y carga computacional. En un principio, se creía que no sería necesaria la programación de un nuevo código, sino que simplemente habría que adaptar el código y optimizar los cuellos de botella detectados en [1], calculando manualmente la detección y discretización de trayectorias. Pero llegados a un punto, se concluyó que la mejor opción sería aprovechar las capacidades del nuevo equipo y reprogramar el código para su uso en la GPU.

5.1.1 Reprogramación del código para la optimización del uso de la GPU

Siendo conscientes de que el equipo sobre el que se trabaja en este TFG se llegó a la conclusión de que explotar el uso de las tarjetas gráficas mejoraría considerablemente los tiempos de ejecución. Es por ello que el grueso del trabajo ha sido la programación del código para adaptarlo a *cuDF* y el uso de la GPU pero manteniendo la extracción de los mismos parámetros que en [1].

Con esta reprogramación se alcanza con creces el objetivo principal, reduciendo en casi cinco veces el tiempo de ejecución en la extracción de parámetros. Esto se obtiene no solo mediante la adaptación del código para que se ejecute sobre la GPU, sino también gracias al cálculo manual de la detección y discretización de las trayectorias evitando el uso de llamadas a funciones de bibliotecas de *movingpandas* y resolviendo el gran coste computacional asumido en TFG previos.

El uso de la GPU reduce los tiempos de ejecución de 73 horas a 16 horas mediante la optimización de los cálculos de los parámetros. Sin embargo, aún no se explota el recurso de la GPU que hace que los tiempos se reduzcan considerablemente: la paralelización. Durante la primera y segunda iteración de este TFG se mantiene el acceso a los datos de forma iterativa mediante el bucle explicado anteriormente y que se ha heredado de TFG previos. Las iteraciones en la GPU provocan que no se explote la paralelización de los procesos ya que se procesarán los datos dinámicos de cada buque secuencialmente y no paralelamente como se pretende para terminar de explotar todo lo que la GPU ofrece.

El motivo por el cual se decide mantener este bucle no es otro que permitir la escalabilidad en el futuro. El procesamiento y extracción de los parámetros se ha realizado para un número cerrado de buques y de mensajes, pero el objetivo de este TFG es que el código desarrollado se pueda usar para tantos buques como sea necesario y el acceso a los datos de forma iterativa permite que el número de

buques sea tan grande como se desee, sin mayor limitación que el tiempo de ejecución aumente de forma proporcional al número de buques y mensajes a procesar.

5.1.2 Extracción de menos parámetros

Tras analizar los resultados obtenidos en los distintos modelos de predicción, se determinó que la mayoría de los parámetros dinámicos no tenían un impacto significativo en la clasificación, a pesar de ser los que más carga computacional requerían. Para optimizar el rendimiento, se procedió a eliminar aquellos parámetros dinámicos con menor relevancia, manteniendo únicamente los 20 más representativos.

Se observó que, al utilizar exclusivamente datos dinámicos para la predicción tras la eliminación de 42 parámetros, la precisión del modelo disminuía considerablemente. Sin embargo, cuando estos se combinaban con los datos estáticos, la calidad de la predicción se mantenía prácticamente inalterada. Tras varias iteraciones de eliminación de columnas, se constató que la inclusión de parámetros dinámicos suponía un alto coste computacional sin aportar una mejora significativa en la predicción en comparación con los datos estáticos. Finalmente, se decidió conservar 20 parámetros dinámicos para que su contribución fuese relevante, aunque los resultados sugieren que este número podría reducirse aún más sin afectar sustancialmente el rendimiento del modelo.

5.1.3 Optimización del acceso a los datos

Tras optimizar los tiempos de ejecución y garantizar la escalabilidad del procesamiento mediante el acceso iterativo a los datos, se exploró la posibilidad de unificar el acceso en un DF único que contuviera toda la información. Una vez implementado este enfoque, se verificó que la eficiencia computacional mejoró drásticamente, reduciendo los tiempos de ejecución a tan solo 10 segundos, lo que permite considerar su aplicación en procesamiento en tiempo real. Sin embargo, esta mejora en velocidad conllevó un incremento significativo en el uso de memoria, ya que la carga de un volumen tan grande de datos en un único DF exige un mayor consumo de recursos, acercándose al límite de la GPU disponible. Esto sugiere que, aunque el modelo actual es eficiente en términos de tiempo de procesamiento, su escalabilidad queda limitada cuando se trabaja con volúmenes de datos aún mayores.

Para abordar este reto, podrían explorarse técnicas de partición y procesamiento por lotes, permitiendo gestionar los datos en segmentos más pequeños sin necesidad de almacenarlos todos simultáneamente en la memoria. Alternativamente, se podría evaluar la integración con *Dask-cuDF*, que permite distribuir la carga en GPU de manera más eficiente, o incluso combinar GPU y CPU en un esquema híbrido para optimizar el uso de los recursos disponibles sin comprometer la velocidad alcanzada.

5.1.4 Predicción del campo *shiptype* mediante algoritmos de aprendizaje automático supervisado

A lo largo de los TFG que preceden a este, se ha usado *Random Forest* como algoritmo de aprendizaje automático supervisado para la predicción del campo *shiptype*. A mayores de los objetivos planteados a comienzos del TFG, se ha decidido implementar el empleo del algoritmo *XGBoost* permitiendo seguir mejorando la precisión en la predicción del *shiptype*. Se ha comprobado que este algoritmo es más potente y que da mejores resultados que el que se venía usando.

El presente TFG se ha realizado íntegramente con bibliotecas que trabajaran en la GPU siendo la principal *cuDF*. Una vez llegado el momento de emplear el algoritmo de aprendizaje automático, se programó con *cuDF* de modo que se ejecutara sobre la GPU, pero al observar cierta pérdida en comparación con la precisión obtenida con *pandas*, se decidió probar a programar el modelo con esta biblioteca sobre la CPU tal y como se ha hecho en los TFG anteriores. Se concluyó que los resultados de los modelos de predicción, tanto *Random Forest* como *XGBoost*, obtenían una mayor precisión al trabajar sobre la CPU.

5.2 Líneas futuras

Considerando las limitaciones expuestas en el apartado anterior, se proponen varias líneas futuras.

5.2.1 Optimización de la gestión de la memoria

Uno de los principales retos identificados en el experimento con mejores resultados es el alto consumo de memoria. Se propone esta línea de trabajo para tratar de mitigar este problema y permitir una mayor escalabilidad mediante estrategias como el procesamiento por lotes (*batch processing*) o el uso de *Dask-cuDF*, que asegura el procesamiento distribuido.

5.2.2 Implementación de un sistema de predicción en tiempo real

Dado los tiempos de ejecución conseguidos, es viable explorar la implementación de este sistema en un contexto de procesamiento en tiempo real. Una posible línea de trabajo futura sería diseñar un sistema que clasifique los buques en tiempo real a medida que llegan los mensajes AIS. Una vez el modelo ha sido entrenado con datos históricos, buscar la inferencia en GPU optimizada, es decir, la clasificación de nuevos ejemplos que el modelo no ha visto antes.

5.2.3 Combinación con imágenes satelitales o radar

Otra posible línea futura sería explorar un sistema multimodal capaz de combinar la información obtenida mediante los datos AIS con imágenes satelitales, radar o de cámaras portuarias. Esta integración permitiría validar y enriquecer los datos AIS aumentando la robustez del sistema frente a manipulaciones de los distintos campos. Además, el uso de imágenes puede aportar variables visuales adicionales que pueden ser útiles para alimentar algoritmos más complejos de aprendizaje profundo.

6 BIBLIOGRAFÍA

- [1] J. Roldán Calvo, «Predicción de tipo de buque utilizando representaciones de trayectorias y técnicas de inteligencia artificial», abr. 2024, Accedido: 24 de febrero de 2025. [En línea]. Disponible en: <http://calderon.cud.uvigo.es/handle/123456789/813>
- [2] R.- ASALE y RAE, «inteligencia | Diccionario de la lengua española», «Diccionario de la lengua española» - Edición del Tricentenario. Accedido: 16 de enero de 2025. [En línea]. Disponible en: <https://dle.rae.es/inteligencia>
- [3] I. de Gandarillas Carrara, «Predicción de tipo de buque utilizando información de áreas de actividad y técnicas de inteligencia artificial», abr. 2023, Accedido: 16 de enero de 2025. [En línea]. Disponible en: <http://calderon.cud.uvigo.es/items/97d3e802-e343-404b-b1cc-b919e657fa34>
- [4] «Historia de la inteligencia artificial», Iberdrola. Accedido: 16 de enero de 2025. [En línea]. Disponible en: <https://www.iberdrola.com/innovacion/historia-inteligencia-artificial>
- [5] «En 1956, John McCarthy acuñó la expresión “inteligencia artificial”, y la definió como “la ciencia e ingenio de hacer máquinas inteligentes” – Empleo UGR. Centro de Empleo y Prácticas.» Accedido: 16 de enero de 2025. [En línea]. Disponible en: <https://empleo.ugr.es/blog-empleo/en-1956-john-mccarthy-acuno-la-expresion-inteligencia-artificial-y-la-definio-como-la-ciencia-e-ingenio-de-hacer-maquinas-inteligentes/>
- [6] «Qué es la Inteligencia Artificial». Accedido: 16 de enero de 2025. [En línea]. Disponible en: <https://planderecuperacion.gob.es/noticias/que-es-inteligencia-artificial-ia-prtr>
- [7] S. Russell, «Inteligencia artificial».
- [8] S. Núñez-Corrales y E. Jakobsson, «*Entropic boundary conditions towards safe artificial superintelligence*», *J. Exp. Theor. Artif. Intell.*, vol. 35, n.º 1, pp. 1-33, ene. 2023, doi: 10.1080/0952813X.2021.1952653.
- [9] «Te contamos qué es el “machine learning” y cómo funciona», BBVA NOTICIAS. Accedido: 19 de enero de 2025. [En línea]. Disponible en: <https://www.bbva.com/es/innovacion/machine-learning-que-es-y-como-funciona/>
- [10] «¿Qué es el machine learning (ML)? | IBM». Accedido: 19 de enero de 2025. [En línea]. Disponible en: <https://www.ibm.com/es-es/topics/machine-learning>
- [11] «*Supervised Learning: From Binary Classification to Polynomial Regression*», Spotfire. Accedido: 19 de enero de 2025. [En línea]. Disponible en: <https://www.spotfire.com/glossary/what-is-supervised-learning>
- [12] G. Rodríguez Casajús, «Predicción de tipo de buque utilizando datos AIS y técnicas de inteligencia artificial», abr. 2022, Accedido: 19 de enero de 2025. [En línea]. Disponible en: <http://calderon.cud.uvigo.es/items/ae14c7df-2065-41af-a83c-bc88ac95d972>
- [13] «¿Qué es el aprendizaje no supervisado y cuándo usarlo?», Inesdi. Accedido: 19 de enero de 2025. [En línea]. Disponible en: <https://www.inesdi.com/blog/que-es-aprendizaje-no-supervisado/>
- [14] J. Luna Gonzalez, «Tipos de aprendizaje automático», SoldAI. Accedido: 20 de enero de 2025. [En línea]. Disponible en: <https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2>

- [15] D. Calvo, «Aprendizaje no supervisado». Accedido: 20 de enero de 2025. [En línea]. Disponible en: <https://www.diegocalvo.es/aprendizaje-no-supervisado/>
- [16] «Qué es Aprendizaje semisupervisado Concepto y definición. Glosario». Accedido: 23 de enero de 2025. [En línea]. Disponible en: <https://gamco.es/glosario/aprendizaje-semisupervisado/>
- [17] «¿Qué es el Aprendizaje mediante refuerzo? - Explicación del Aprendizaje mediante refuerzo - AWS», Amazon Web Services, Inc. Accedido: 23 de enero de 2025. [En línea]. Disponible en: <https://aws.amazon.com/es/what-is/reinforcement-learning/>
- [18] «Aprendizaje por Refuerzo | Aprende Machine Learning». Accedido: 26 de enero de 2025. [En línea]. Disponible en: <https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/>
- [19] M. Silva, «Aprendizaje por Refuerzo: Procesos de Decisión de Markov — Parte 1», Aprendizaje por Refuerzo: Introducción al mundo del RL. Accedido: 26 de enero de 2025. [En línea]. Disponible en: <https://medium.com/aprendizaje-por-refuerzo-introducci%C3%B3n-al-mundo-del/aprendizaje-por-refuerzo-procesos-de-decisi%C3%B3n-de-markov-parte-1-8a0aed1e6c59>
- [20] J. D. Kelleher, *Deep Learning*. MIT Press, 2019.
- [21] I. Pérez Borrero y M. E. García Arias, *DEEP LEARNING*. Servicio de Publicaciones de la Universidad de Huelva, 2021.
- [22] «¿Qué es el deep learning? | IBM». Accedido: 27 de enero de 2025. [En línea]. Disponible en: <https://www.ibm.com/es-es/topics/deep-learning>
- [23] «*Machine Learning Examples, Applications & Use Cases* | IBM». Accedido: 29 de enero de 2025. [En línea]. Disponible en: <https://www.ibm.com/think/topics/machine-learning-use-cases>
- [24] «¿Qué es un árbol de decisión? | IBM». Accedido: 28 de enero de 2025. [En línea]. Disponible en: <https://www.ibm.com/es-es/topics/decision-trees>
- [25] «Qué son los árboles de decisión y para qué sirven», Máxima Formación. Accedido: 28 de enero de 2025. [En línea]. Disponible en: <https://www.maximaformacion.es/blog-dat/que-son-los-arboles-de-decision-y-para-que-sirven/>
- [26] «Árboles de decisión: qué son, para qué sirven y ejemplo». Accedido: 28 de enero de 2025. [En línea]. Disponible en: <https://www.educaopen.com/digital-lab/blog/inteligencia-artificial/arboles-de-decision>
- [27] «*Introduction to Random Forest in Machine Learning*», www.cloudzilla.ai. Accedido: 29 de enero de 2025. [En línea]. Disponible en: <https://www.cloudzilla.ai/dev-education/introduction-to-random-forest-in-machine-learning/>
- [28] «¿Qué es el sobreajuste? - Explicación del sobreajuste en machine learning - AWS», Amazon Web Services, Inc. Accedido: 29 de enero de 2025. [En línea]. Disponible en: <https://aws.amazon.com/es/what-is/overfitting/>
- [29] T. Yiu, «*Understanding Random Forest*», Medium. Accedido: 29 de enero de 2025. [En línea]. Disponible en: <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
- [30] Sruthi, «*Understanding Random Forest Algorithm With Examples*», Analytics Vidhya. Accedido: 29 de enero de 2025. [En línea]. Disponible en: <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>
- [31] «Cómo funciona el algoritmo XGBoost». Accedido: 28 de enero de 2025. [En línea]. Disponible en: <https://pro.arcgis.com/es/pro-app/latest/tool-reference/geoai/how-xgboost-works.htm>
- [32] «*Extreme Gradient Boosting (XGBOOST)*», XLSTAT, Your data analysis solution. Accedido: 28 de enero de 2025. [En línea]. Disponible en: <https://www.xlstat.com/es/soluciones/funciones/extreme-gradient-boosting-xgboost>
- [33] «¿Qué es XGBoost? | IBM». Accedido: 28 de enero de 2025. [En línea]. Disponible en: <https://www.ibm.com/es-es/topics/xgboost>
- [34] L. Álamo, «La Armada usará inteligencia artificial para vigilar el entorno marítimo desde Cartagena», La Opinión de Murcia. Accedido: 6 de febrero de 2025. [En línea]. Disponible en: <https://www.laopiniondemurcia.es/cartagena/2021/11/10/armada-usara-inteligencia-artificial-vigilar-59387366.html>

- [35] A.-L. S. Lapinski, «*A strategy for uncertainty visualization design*», *DRDC Atl. TM 2009*, vol. 151, 2009.
- [36] Defensa.com, «Centro de operaciones y vigilancia de acción marítima de la Armada - Noticias Defensa En abierto», Defensa.com. Accedido: 6 de febrero de 2025. [En línea]. Disponible en: <https://www.defensa.com/en-abierto/centro-operaciones-vigilancia-accion-maritima-armada>
- [37] «Estrategia Nacional de Seguridad Marítima 2024», 2024.
- [38] Armada, «El COVAM de la Armada al servicio de la Comunidad Marítima - El COVAM de la Armada al servicio de la Comunidad Marítima - Especiales - Armada - Ministerio de Defensa - Gobierno de España». Accedido: 6 de febrero de 2025. [En línea]. Disponible en: <https://armada.defensa.gob.es/ArmadaPortal/page/Portal/ArmadaEspañola/conocenosespeciales/prefLang-es/05actividades--seguridadmaritima--02COVAMIEE>
- [39] «▷ Sistema AIS - ¿Qué es?» Accedido: 6 de febrero de 2025. [En línea]. Disponible en: <https://www.boletinpatron.com/el-sistema-de-identificacion-automatica-de-buques-ais/>
- [40] S. Bhattacharjee, «*What is Automatic Identification System (AIS)- Types and Working (FAQs)*», Marine Insight. Accedido: 6 de febrero de 2025. [En línea]. Disponible en: <https://www.marineinsight.com/marine-navigation/automatic-identification-system-ais-integrating-and-identifying-marine-communication-channels/>
- [41] «El país fronterizo de la OTAN da el aviso tras ver lo que navega hacia Rusia: “Pone en peligro la seguridad”», ElHuffPost. Accedido: 6 de febrero de 2025. [En línea]. Disponible en: <https://www.huffingtonpost.es/global/el-pais-fronterizo-otan-da-aviso-ver-navega-rusia-pone-peligro-seguridad.html>
- [42] R. Fernández Martín «Validación de una serie de parámetros estáticos y dinámicos para la predicción de tipo de buque utilizando técnicas de inteligencia artificial». Accedido: 25 de marzo de 2025. [En línea]. Disponible en: <http://calderon.cud.uvigo.es/bitstreams/5305de36-d615-432d-b415-af7ec12a9abd/download>
- [43] «*Welcome to the cudf documentation! — cudf 25.02.00 documentation*». Accedido: 25 de febrero de 2025. [En línea]. Disponible en: <https://docs.rapids.ai/api/cudf/stable/>
- [44] «*RAPIDS - Get Started*», NVIDIA Developer. Accedido: 25 de febrero de 2025. [En línea]. Disponible en: <https://developer.nvidia.com/rapids/get-started>
- [45] «La GPU NVIDIA acelera la ciencia de datos con RAPIDS», NVIDIA. Accedido: 10 de febrero de 2025. [En línea]. Disponible en: <https://www.nvidia.com/es-es/deep-learning-ai/software/rapids/>
- [46] «*Project Jupyter*». Accedido: 4 de marzo de 2025. [En línea]. Disponible en: <https://jupyter.org>
- [47] «*Welcome to Python.org*», Python.org. Accedido: 25 de febrero de 2025. [En línea]. Disponible en: <https://www.python.org/>
- [48] «*NumPy*». Accedido: 19 de marzo de 2025. [En línea]. Disponible en: <https://numpy.org/>
- [49] «*pandas - Python Data Analysis Library*». Accedido: 19 de marzo de 2025. [En línea]. Disponible en: <https://pandas.pydata.org/>
- [50] J. Ali, R. Khan, N. Ahmad, y I. Maqsood, «*Random Forests and Decision Trees*», vol. 9, n.º 5, 2012.
- [51] «*XGBoost Parameters — xgboost 3.0.0 documentation*». Accedido: 19 de marzo de 2025. [En línea]. Disponible en: <https://xgboost.readthedocs.io/en/stable/parameter.html>
- [52] «*RAPIDS Suite of AI Libraries*», NVIDIA Developer. Accedido: 25 de febrero de 2025. [En línea]. Disponible en: <https://developer.nvidia.com/rapids>
- [53] A. Gupta, «*XGBoost versus Random Forest*», Geek Culture. Accedido: 25 de febrero de 2025. [En línea]. Disponible en: <https://medium.com/geekculture/xgboost-versus-random-forest-898e42870f30>
- [54] S. V. B. GmbH, «Todo lo que debe saber sobre AIS», SVB GmbH. Accedido: 25 de febrero de 2025. [En línea]. Disponible en: <https://www.svb-marine.es/es/guia/todo-lo-que-debe-saber-sobre-ais.html>

- [55] J. B. Mendoza Vega, «Tutorial: XGBoost en Python», Medium. Accedido: 25 de febrero de 2025. [En línea]. Disponible en: <https://medium.com/@jboscomendoza/tutorial-xgboost-en-python-53e48fc58f73>
- [56] D Kirk, & W. M. HWU, *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann. Burlington, MA, USA: Morgan Kaufmann (imprint of Elsevier), 2010.
- [57] «cuDF Pandas», RAPIDS | GPU Accelerated Data Science. Accedido: 28 de febrero de 2025. [En línea]. Disponible en: <https://rapids.ai/cudf-pandas/>
- [58] «¿Qué es la fórmula Haversine?» Accedido: 28 de febrero de 2025. [En línea]. Disponible en: <https://es.locationiq.com/glossary/haversine-formula>
- [59] T. Hastie, R. Tibshirani, y J. Friedman, *The Elements of Statistical Learning*, Segunda edición. en Springer Series in Statistics. Springer.
- [60] «Floating Point and IEEE 754 12.8 documentation». Accedido: 11 de marzo de 2025. [En línea]. Disponible en: <https://docs.nvidia.com/cuda/floating-point/index.html>
- [61] «pandas: a Foundational Python Library for Data Analysis and Statistics», ResearchGate. Accedido: 11 de marzo de 2025. [En línea]. Disponible en: https://www.researchgate.net/publication/265194455_pandas_a_Foundational_Python_Library_for_Data_Analysis_and_Statistics
- [62] E. A. Pérez Tatamués, TFG «Algoritmo de Random Forest aplicado a la detección de fraude en el sistema bancario ecuatoriano».

ANEXO I: IMPLICACIONES SOCIALES, ECONÓMICAS Y AMBIENTALES

Siguiendo los criterios del Sello Internacional de Calidad EUR-ACE, este anexo recoge un análisis de los impactos sociales, económicos y ambientales vinculados a este TFG. A lo largo de la memoria se han abordado diferentes secciones que reflejan la importancia de estos aspectos. El objetivo de este anexo es sintetizar dichas consideraciones y destacar la influencia que este trabajo puede tener en distintos ámbitos.

El desarrollo de un sistema de predicción de tipo de buque basado en inteligencia artificial puede contribuir a la mejora de la seguridad marítima y la eficiencia en la gestión del tráfico marítimo. La capacidad de identificar y clasificar buques en tiempo real puede ayudar a optimizar las rutas de navegación, reducir riesgos de colisión y mejorar la coordinación en puertos y terminales. Asimismo, este tipo de tecnología puede apoyar a organismos reguladores en la supervisión del cumplimiento normativo y la prevención de actividades ilícitas en el mar.

El uso de inteligencia artificial requiere infraestructuras computacionales potentes, como GPU y centros de datos, que consumen grandes cantidades de energía. Este elevado consumo contribuye a la huella de carbono, especialmente si la electricidad proviene de fuentes no renovables. Por ello, es importante diseñar soluciones eficientes también desde el punto de vista energético. En este trabajo, las optimizaciones aplicadas para reducir los tiempos de ejecución contribuyen a minimizar el impacto ambiental, favoreciendo un uso más sostenible de la inteligencia artificial.

Desde una perspectiva económica, la mejora en la predicción y clasificación de buques puede incrementar la eficiencia operativa en los puertos, reduciendo costes asociados a retrasos y congestión en las terminales. La implementación de modelos de inteligencia artificial optimizados para GPU, como se ha planteado en este trabajo, permite una reducción significativa en los tiempos de procesamiento, lo que se traduce en una mayor capacidad de respuesta en aplicaciones comerciales y logísticas. Adicionalmente, la optimización del uso de recursos en el sector marítimo puede generar un impacto positivo en la competitividad de las empresas del sector.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

El desarrollo de tecnologías de inteligencia artificial en el ámbito marítimo plantea serias cuestiones éticas y sociales que deben ser abordadas responsablemente. En este anexo, se tienen en consideración algunos de los principales problemas y dilemas inherentes a la aplicación de sistemas de predicción automáticos.

Uno de los principales dilemas éticos en el uso de inteligencia artificial para el pronóstico de clases de buques es asegurar que los modelos y algoritmos utilizados sean transparentes. Dado que estas tecnologías pueden tener un efecto en decisiones operativas y regulatorias, es fundamental que sus procesos sean comprensibles para los usuarios y que se establezcan mecanismos de auditoría para determinar su fiabilidad y exactitud.

La automatización de procesos mediante inteligencia artificial puede causar transformaciones en el ámbito laboral, especialmente en industrias en las que el proceso de toma de decisiones históricamente se había basado en la experiencia humana. Si bien estas tecnologías pueden mejorar el rendimiento y eliminar errores humanos, también deben considerarse estrategias de readaptación para empleados que queden afectados por la digitalización, promoviendo la capacitación en nuevas competencias tecnológicas.

La creación de modelos de predicción depende en gran medida de tener grandes cantidades de datos. Por eso, es necesario asegurarse de que la recopilación y uso de los datos estén en armonía con los reglamentos de privacidad e información. Por otro lado, se deben establecer medidas para impedir sesgos en los modelos que puedan llevar a decisiones injustas o discriminatorias.

La aplicación de la inteligencia artificial en el ámbito marítimo, además de los asuntos relacionados con la protección de datos, puede ser problemática si las tecnologías son utilizadas con fines maliciosos. Es fundamental establecer protocolos de seguridad para evitar que estos sistemas se usen para fines ilícitos, como el contrabando ilegal o la evasión de reglamentos. La cooperación entre instituciones públicas y privadas puede ser decisiva para afrontar estos riesgos y asegurar un uso seguro de la tecnología.

Por tanto, el empleo de inteligencia artificial en el ámbito naval debe ir acompañado de un enfoque ético y socialmente responsable para asegurar que sus beneficios se maximicen sin comprometer principios esenciales de equidad, transparencia y seguridad.

ANEXO III: CÓDIGO DESARROLLADO

```
In [1]: import torch
print(torch.cuda.current_device())
torch.cuda.set_device(1)
print(torch.cuda.current_device())

0
1

In [2]: import csv
# Para leer y escribir archivos CSV
import os
# Para manipular archivos y directorios
import pandas as pd
# Para manipulación y análisis de datos
import geopandas as gpd
# Para trabajar con datos geoespaciales
import movingpandas as mpd
# Para el análisis de datos de movimiento
import numpy as np
# Para computación científica y operaciones matemáticas
from tqdm import tqdm
# Para mostrar barras de progreso en bucles iterativos
import seaborn as sns
# Para visualización de datos estadísticos
import matplotlib.pyplot as plt
# Para visualización de datos en gráficos
from sklearn.model_selection import train_test_split, GridSearchCV, GroupShuffleSplit
# Para dividir conjuntos de datos y búsqueda de hiperparámetros
from sklearn.preprocessing import MinMaxScaler
# Para escalar características
from sklearn.metrics import classification_report
# Para evaluar el rendimiento del modelo
from sklearn.ensemble import RandomForestClassifier
# Para entrenar modelos de clasificación de bosques aleatorios
from sklearn.pipeline import Pipeline
# Para encadenar pasos de procesamiento y modelado
import hvplot.pandas
# Para visualización interactiva de pandas DataFrames
from shapely.geometry import Point
# Para trabajar con geometría en Python
from datetime import timedelta
# Para manipular fechas y tiempos
import geoviews as gv
# Para visualización de datos geoespaciales
from ydata_profiling import ProfileReport
# Para generar informes de perfil de datos
import holoviews as hv
# Para visualización interactiva de datos
from geopy.distance import great_circle, geodesic
# Para calcular distancias geoespaciales
from shapely.geometry import MultiPoint, LineString, Polygon
# Para trabajar con geometría en Python
from matplotlib.colors import LinearSegmentedColormap
# Para personalizar mapas de colores en Matplotlib
import xarray as xr
# Para trabajar con estructuras de datos multidimensionales
import cartopy.crs as ccrs
# Para visualización de mapas utilizando Cartopy
import plotly.express as px
# Para visualización interactiva de datos
import time
# Para medir el tiempo de ejecución de código

# Adaptación para GPU
import cudf
import cupy as cp
import hvplot.cudf

In [3]: # Obtiene la lista de archivos CSV en el directorio
lista_archivos_dynamic = (
    [archivo for archivo in os.listdir('Dynamic_by_MMSI_month') if archivo.endswith('.csv')])
# Lee el primer archivo CSV y lo almacena en un DataFrame
df_raw_dynamic = pd.read_csv(os.path.join('Dynamic_by_MMSI_month', lista_archivos_dynamic[0]))
df_raw_dynamic.head(1)

Out[3]: Unnamed: 0  timestamp      mmsi      lat      lon  speed  course  heading  shiptype
0          101  1674428298  232018697 -10.14435 -34.270712   11.4    26.2     25  ShipType.Cargo

In [4]: df_MMSI_shiptype = (
    pd.read_csv('final_static_data_jan.csv')
    .filter(items=['MMSI', 'shiptype'])
    .sort_values(by='MMSI')
    .reset_index(drop=True)
)
df_MMSI_shiptype.to_csv('MMSI_shiptype_data.csv', index = False)

In [5]: print (df_MMSI_shiptype)
```

```

      MMSI  shiptype
0         5        4
1         6        4
2       2829        0
3       3247        2
4       8086        2
...      ...      ...
40854  827002226    4
40855  848290500    0
40856  900004001    0
40857  924467387    1
40858  985524411    4

[40859 rows x 2 columns]

In [6]: # Realiza el merge basado en las columnas 'mmsi' y 'MMSI'
df_merged_dynamic = pd.merge(df_raw_dynamic, df_MMSI_shiptype, left_on='mmsi', right_on='MMSI',
                             how='inner')
# Elimina la columna 'MMSI' duplicada
df_merged_dynamic.drop(columns=['MMSI'], inplace=True)
# Elimina la columna 'shiptype_y'
df_merged_dynamic.drop(columns=['shiptype_y'], inplace=True)
# Cambia el nombre de 'ship_type_x' a 'shiptype'
df_merged_dynamic.rename(columns={'shiptype_x': 'shiptype'}, inplace=True)
# Elimina filas con valores NaN en la columna 'shiptype'
df_merged_dynamic.dropna(subset=['shiptype'], inplace=True)
# Ordena por timestamp
df_merged_dynamic = df_merged_dynamic.sort_values(by='timestamp')
df_merged_dynamic = df_merged_dynamic.drop(columns=['Unnamed: 0'])

df_merged_dynamic.replace([np.inf, -np.inf], np.nan, inplace=True)
df_merged_dynamic = df_merged_dynamic.dropna()

print(df_merged_dynamic)

   timestamp  mmsi      lat      lon  speed  course  heading \
486  1672531377  232018697  49.339297 -4.751987    7.3   251.5   246
487  1672531706  232018697  49.335242 -4.767602    7.2   245.8   239
488  1672532057  232018697  49.329048 -4.782570    6.8   235.6   236
489  1672533097  232018697  49.309785 -4.827603    7.4   236.7   236
490  1672533467  232018697  49.302508 -4.844132    7.7   234.6   236
...      ...      ...      ...      ...      ...      ...
2186  1675204884  232018697  24.987058 -17.793093   11.0   24.9   25
2187  1675205333  232018697  25.013247 -17.780293   11.0   22.3   24
1658  1675205790  232018697  25.046707 -17.764345   10.8   24.0   24
1659  1675208152  232018697  25.152068 -17.711468   10.7   24.1   25
1660  1675209285  232018697  25.125102 -17.725257   10.7   25.1   26

      shiptype
486  ShipType.Cargo
487  ShipType.Cargo
488  ShipType.Cargo
489  ShipType.Cargo
490  ShipType.Cargo
...      ...
2186  ShipType.Cargo
2187  ShipType.Cargo
1658  ShipType.Cargo
1659  ShipType.Cargo
1660  ShipType.Cargo

[2286 rows x 8 columns]

In [7]: print(df_merged_dynamic.isna().any().any())
df_merged_dynamic.dtypes

False

Out[7]: timestamp    int64
mmsi              int64
lat              float64
lon              float64
speed            float64
course           float64
heading          int64
shiptype         object
dtype: object

In [8]: # Inicializa un DataFrame para almacenar las longitudes según el shiptype
data = {'shiptype': [], 'longitud': []}

for MMSI, row in tqdm(df_MMSI_shiptype.iterrows()):
    # Busca el archivo CSV correspondiente al MMSI actual
    csv_file = os.path.join('Dynamic_by_MMSI_month', f"{row['MMSI']}.csv")

    # Verifica si el archivo CSV existe
    if os.path.exists(csv_file):
        df_raw_dynamic = pd.read_csv(csv_file)

        # Calcula la longitud del DataFrame actual y añádela al DataFrame de datos
        shiptype = row['shiptype']
        longitud_df = len(df_raw_dynamic)
        data['shiptype'].append(shiptype)
        data['longitud'].append(longitud_df)

# Crea un DataFrame a partir de los datos recopilados
df_longitudes = pd.DataFrame(data)

40859it [01:05, 619.10it/s]

In [10]: df_longitudes

```

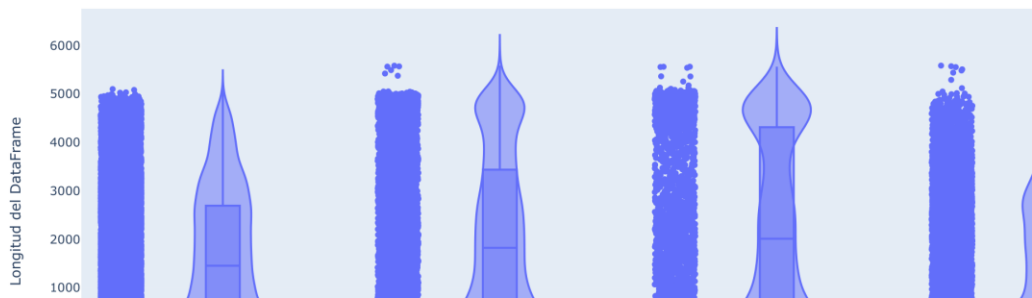
```
Out[10]:
```

	shiptype	longitud
0	4	4416
1	4	4701
2	0	2522
3	2	1065
4	2	1311
...	...	...
40680	4	1619
40681	0	2860
40682	0	1795
40683	1	3998
40684	4	627

40685 rows × 2 columns

```
In [11]: # Crea el violinplot para mostrar las distribuciones de Longitudes según el shiptype,
# usando px (interactivo)
fig = px.violin(df_longitudes, y='longitud', x='shiptype', box=True, points="all")
fig.update_layout(title='Distribución de longitudes según el shiptype',
                  xaxis_title='Shiptype',
                  yaxis_title='Longitud del DataFrame')
fig.show()
```

Distribución de longitudes según el shiptype



```
In [8]: %%time
# Función para calcular distancia entre coordenadas con Haversine en GPU
def haversine_gpu(lat1, lon1, lat2, lon2):
    R = 6371000 # Radio de La Tierra en metros
    phi1 = cp.radians(lat1)
    phi2 = cp.radians(lat2)
    delta_phi = cp.radians(lat2 - lat1)
    delta_lambda = cp.radians(lon2 - lon1)

    a = cp.sin(delta_phi / 2.0) ** 2 + cp.cos(phi1) * cp.cos(phi2) * cp.sin(delta_lambda / 2.0) ** 2
    c = 2 * cp.arctan2(cp.sqrt(a), cp.sqrt(1 - a))
    return R * c

# Función optimizada para calcular trayectorias en cuDF
def calcular_trayectorias(df):
    df = df.sort_values(by=['mmsi', 'timestamp']) # Ordenar datos en GPU

    df['time_diff'] = df['timestamp'].diff() # Calcular el tiempo entre puntos (en segundos)

    # Calcular distancia entre puntos consecutivos usando CuPy
    df['lat_shift'] = df.groupby('mmsi')['lat'].shift()
    df['lon_shift'] = df.groupby('mmsi')['lon'].shift()
    df = df.dropna()

    lat1 = cp.asarray(df['lat'].values)
    lon1 = cp.asarray(df['lon'].values)
    lat2 = cp.asarray(df['lat_shift'].values)
    lon2 = cp.asarray(df['lon_shift'].values)
    df['distance'] = haversine_gpu(lat1, lon1, lat2, lon2).get()

    # Crear una nueva columna para detectar paradas
    df['is_stop'] = ((df['distance'] < 50) & (df['time_diff'] > 3600)).astype("int32")

    # Crear identificador acumulativo de trayectorias Asegurar que la segmentación se actualiza solo cuando cambia de 0 a 1
    df['trajectory_id'] = (df['is_stop'].shift(fill_value=0) == 0) & (df['is_stop'] == 1)
```

```

df['trajectory_id'] = df['trajectory_id'].cumsum()

return df

```

CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 8.11 µs

In [9]: df_merged_dynamic

Out[9]:

	timestamp	mmsi	lat	lon	speed	course	heading	shiptype
486	1672531377	232018697	49.339297	-4.751987	7.3	251.5	246	ShipType.Cargo
487	1672531706	232018697	49.335242	-4.767602	7.2	245.8	239	ShipType.Cargo
488	1672532057	232018697	49.329048	-4.782570	6.8	235.6	236	ShipType.Cargo
489	1672533097	232018697	49.309785	-4.827603	7.4	236.7	236	ShipType.Cargo
490	1672533467	232018697	49.302508	-4.844132	7.7	234.6	236	ShipType.Cargo
...	...	...	...	...	...	...	...	...
2186	1675204884	232018697	24.987058	-17.793093	11.0	24.9	25	ShipType.Cargo
2187	1675205333	232018697	25.013247	-17.780293	11.0	22.3	24	ShipType.Cargo
1658	1675205790	232018697	25.046707	-17.764345	10.8	24.0	24	ShipType.Cargo
1659	1675208152	232018697	25.152068	-17.711468	10.7	24.1	25	ShipType.Cargo
1660	1675209285	232018697	25.125102	-17.725257	10.7	25.1	26	ShipType.Cargo

2286 rows × 8 columns

In [10]: df_merged_dynamic.isna().any().any()

Out[10]: np.False_

In [11]: month_trajectories = calcular_trayectorias(df_merged_dynamic)
month_trajectories.to_csv('prueba_month_trajectories.csv')

In [12]: month_trajectories

Out[12]:

	timestamp	mmsi	lat	lon	speed	course	heading	shiptype	time_diff	lat_shift	lon_shift	distance	is_stop	trajector
487	1672531706	232018697	49.335242	-4.767602	7.2	245.8	239	ShipType.Cargo	329.0	49.339297	-4.751987	1217.926426	0	
488	1672532057	232018697	49.329048	-4.782570	6.8	235.6	236	ShipType.Cargo	351.0	49.335242	-4.767602	1284.822855	0	
489	1672533097	232018697	49.309785	-4.827603	7.4	236.7	236	ShipType.Cargo	1040.0	49.329048	-4.782570	3904.101997	0	
490	1672533467	232018697	49.302508	-4.844132	7.7	234.6	236	ShipType.Cargo	370.0	49.309785	-4.827603	1445.972581	0	
491	1672533857	232018697	49.294863	-4.861235	7.3	234.8	236	ShipType.Cargo	390.0	49.302508	-4.844132	1503.553099	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2186	1675204884	232018697	24.987058	-17.793093	11.0	24.9	25	ShipType.Cargo	1288.0	24.865445	-17.854253	14862.675550	0	
2187	1675205333	232018697	25.013247	-17.780293	11.0	22.3	24	ShipType.Cargo	449.0	24.987058	-17.793093	3184.993348	0	
1658	1675205790	232018697	25.046707	-17.764345	10.8	24.0	24	ShipType.Cargo	457.0	25.013247	-17.780293	4052.718494	0	
1659	1675208152	232018697	25.152068	-17.711468	10.7	24.1	25	ShipType.Cargo	2362.0	25.046707	-17.764345	12868.773282	0	
1660	1675209285	232018697	25.125102	-17.725257	10.7	25.1	26	ShipType.Cargo	1133.0	25.152068	-17.711468	3304.171975	0	

2285 rows × 14 columns

In [13]: # Función para detectar paradas en cuDF (sin MovingPandas)
def detectar_paradas_cudf(df):
 df = df.sort_values(by=['mmsi', 'timestamp']) # Ordenar datos por barco y tiempo

 # Calcular tiempo entre mensajes
 df['time_diff'] = df.groupby('mmsi')['timestamp'].diff().dt.total_seconds()

 # Filtrar puntos de parada (tiempo > 3600s y distancia < 50m)
 stop_points = df[(df['distance'] < 50) & (df['time_diff'] > 3600)]

 return stop_points[['mmsi', 'timestamp', 'lat', 'lon', 'distance', 'time_diff']]

In [14]: # Función optimizada para dividir trayectorias en segmentos separados por paradas
def dividir_trayectorias_por_paradas(df):
 df['is_stop'] = ((df['distance'] < 50) & (df['time_diff'] > 3600)).astype("int32")

 # Asegurar que la segmentación se actualiza solo cuando cambia de 0 a 1
 df['split_id'] = (df['is_stop'].shift(fill_value=0) == 0) & (df['is_stop'] == 1)
 df['split_id'] = df['split_id'].cumsum()
 return df[['mmsi', 'timestamp', 'lat', 'lon', 'speed', 'distance', 'time_diff', 'split_id',]]
month_separated_trajectories = dividir_trayectorias_por_paradas(month_trajectories)
print(month_separated_trajectories)

MEJORA DE LA EFICIENCIA COMPUTACIONAL DE UN SISTEMA DE PREDICCIÓN DE TIPO DE BUQUE UTILIZANDO PARÁMETROS DE TRAYECTORIAS Y TÉCNICAS DE IA

	mmsi	timestamp	lat	lon	speed	distance \
487	232018697	1672531706	49.335242	-4.767602	7.2	1217.926426
488	232018697	1672532057	49.329048	-4.782570	6.8	1284.822855
489	232018697	1672533097	49.309785	-4.827603	7.4	3904.101997
490	232018697	1672533467	49.302508	-4.844132	7.7	1445.972581
491	232018697	1672533857	49.294863	-4.861235	7.3	1503.553099
...	...	...	...	...	...	...
2186	232018697	1675204884	24.987058	-17.793093	11.0	14862.675550
2187	232018697	1675205333	25.013247	-17.780293	11.0	3184.993348
1658	232018697	1675205790	25.046707	-17.764345	10.8	4052.718494
1659	232018697	1675208152	25.152068	-17.711468	10.7	12868.773282
1660	232018697	1675209285	25.125102	-17.725257	10.7	3304.171975

	time_diff	split_id
487	329.0	0
488	351.0	0
489	1040.0	0
490	370.0	0
491	390.0	0
...	...	...
2186	1288.0	8
2187	449.0	8
1658	457.0	8
1659	2362.0	8
1660	1133.0	8

[2285 rows x 8 columns]

```
In [15]: # Función optimizada para detectar segmentos de paradas en cuDF
def detectar_segmentos_parada_cudf(df):
    df = df.sort_values(by='timestamp') # Ordenar solo por tiempo

    # Calcular diferencia de tiempo en segundos
    df['time_diff'] = df['timestamp'].diff().fillna(0).astype(int)

    # Identificar si es una parada (distancia < 50m y tiempo > 3600s)
    df['is_stop'] = ((df['distance'] < 50) & (df['time_diff'] > 3600)).astype("int32")

    # Crear identificador de segmento de parada
    df['split_id'] = ((df['is_stop'].shift(fill_value=0) == 0) & (df['is_stop'] == 1)).cumsum()

    # Filtrar solo los segmentos de parada
    mask = df['is_stop'] == 1
    return df.loc[mask, ['mmsi', 'timestamp', 'lat', 'lon', 'distance', 'time_diff', 'split_id']]
```

```
In [16]: # Cargar un archivo CSV específico en cuDF para pruebas
ejemplo_raw = cudf.read_csv(os.path.join('Dynamic_by_MMSI_month', lista_archivos_dynamic[243]))

# Verificar y convertir tipos antes de la fusión
df_MMSI_shiptype = cudf.DataFrame(df_MMSI_shiptype).reset_index() # Asegurar cuDF y resetear índice
df_MMSI_shiptype['MMSI'] = df_MMSI_shiptype['MMSI'].astype('int64')
ejemplo_raw['mmsi'] = ejemplo_raw['mmsi'].astype('int64')

# Realizar merge en cuDF
ejemplo_merged = ejemplo_raw.merge(df_MMSI_shiptype, left_on='mmsi', right_on='MMSI', how='inner')
ejemplo_merged = ejemplo_merged.sort_values(by='timestamp')

# Limpieza de columnas
ejemplo_merged = ejemplo_merged.drop(columns=['MMSI'], errors='ignore')
ejemplo_merged = ejemplo_merged.drop(columns=['shiptype_y'], errors='ignore')
ejemplo_merged = ejemplo_merged.rename(columns={'shiptype_x': 'shiptype'})
ejemplo_merged = ejemplo_merged.dropna(subset=['shiptype']) # Eliminar NaN en shiptype

# Convertir timestamp a formato datetime en cuDF
ejemplo_merged['timestamp'] = cudf.to_datetime(ejemplo_merged['timestamp'], unit='s')
print(ejemplo_merged) # Imprimir resultado para verificar

# Fusionar con el DataFrame de MMSI y shiptype
ejemplo_raw['mmsi'] = ejemplo_raw['mmsi'].astype('int64')
df_MMSI_shiptype['MMSI'] = df_MMSI_shiptype['MMSI'].astype('int64')
ejemplo_merged = ejemplo_raw.merge(df_MMSI_shiptype, left_on='mmsi', right_on='MMSI', how='inner')
ejemplo_merged = ejemplo_merged.sort_values(by='timestamp')
ejemplo_merged = ejemplo_merged.drop(columns=['MMSI'])
ejemplo_merged = ejemplo_merged.drop(columns=['shiptype_y'])
ejemplo_merged = ejemplo_merged.rename(columns={'shiptype_x': 'shiptype'})
ejemplo_merged = ejemplo_merged.dropna(subset=['shiptype'])
ejemplo_merged['timestamp'] = cudf.to_datetime(ejemplo_merged['timestamp'], unit='s')

print(ejemplo_merged)
```

```

      Unnamed: 0      timestamp      mmsi      lat      lon      speed \
150      814 2023-01-19 04:14:54 357144000 34.023170 127.809328 12.3
156      815 2023-01-19 04:33:43 357144000 34.071153 127.862763 12.7
159      816 2023-01-19 04:39:11 357144000 34.085202 127.878988 12.7
179      817 2023-01-19 04:45:02 357144000 34.100042 127.896383 12.8
223      818 2023-01-19 04:50:12 357144000 34.113212 127.911872 12.9
..      ...      ...      ...      ...      ...      ...
214      2050 2023-01-31 23:11:49 357144000 -4.986465 153.772437 14.4
261      2051 2023-01-31 23:27:53 357144000 -5.042668 153.793877 14.3
265      2052 2023-01-31 23:38:32 357144000 -5.092223 153.813010 14.2
297      2053 2023-01-31 23:47:02 357144000 -5.118488 153.822072 14.2
306      2054 2023-01-31 23:56:48 357144000 -5.141020 153.829672 14.2

```

```

      course      heading      shiptype      index
150      40.0      38      ShipType.Tanker      18873
156      43.5      42      ShipType.Tanker      18873
159      44.2      43      ShipType.Tanker      18873
179      44.7      42      ShipType.Tanker      18873
223      44.7      43      ShipType.Tanker      18873
..      ...      ...      ...      ...
214      157.3      157      ShipType.Tanker      18873
261      158.2      157      ShipType.Tanker      18873
265      160.3      159      ShipType.Tanker      18873
297      160.5      159      ShipType.Tanker      18873
306      162.3      159      ShipType.Tanker      18873

```

```

[642 rows x 10 columns]
      Unnamed: 0      timestamp      mmsi      lat      lon      speed \
333      814 2023-01-19 04:14:54 357144000 34.023170 127.809328 12.3
343      815 2023-01-19 04:33:43 357144000 34.071153 127.862763 12.7
346      816 2023-01-19 04:39:11 357144000 34.085202 127.878988 12.7
337      817 2023-01-19 04:45:02 357144000 34.100042 127.896383 12.8
350      818 2023-01-19 04:50:12 357144000 34.113212 127.911872 12.9
..      ...      ...      ...      ...      ...      ...
235      2050 2023-01-31 23:11:49 357144000 -4.986465 153.772437 14.4
364      2051 2023-01-31 23:27:53 357144000 -5.042668 153.793877 14.3
366      2052 2023-01-31 23:38:32 357144000 -5.092223 153.813010 14.2
368      2053 2023-01-31 23:47:02 357144000 -5.118488 153.822072 14.2
378      2054 2023-01-31 23:56:48 357144000 -5.141020 153.829672 14.2

```

```

      course      heading      shiptype      index
333      40.0      38      ShipType.Tanker      18873
343      43.5      42      ShipType.Tanker      18873
346      44.2      43      ShipType.Tanker      18873
337      44.7      42      ShipType.Tanker      18873
350      44.7      43      ShipType.Tanker      18873
..      ...      ...      ...      ...
235      157.3      157      ShipType.Tanker      18873
364      158.2      157      ShipType.Tanker      18873
366      160.3      159      ShipType.Tanker      18873
368      160.5      159      ShipType.Tanker      18873
378      162.3      159      ShipType.Tanker      18873

```

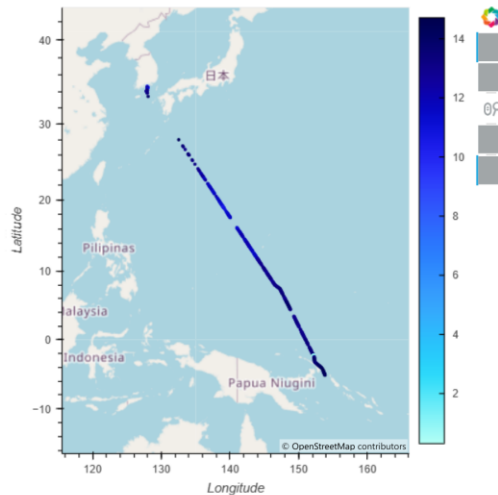
```
[642 rows x 10 columns]
```

```

In [17]: # Visualización optimizada en GPU con hvplot
ejemplo_merged.hvplot.points(x='lon', y='lat', geo=True, tiles=True, width=500, height=500, size=5, color='speed', colorbar=True)

```

```
Out[17]:
```



```

In [18]: # Aplicar funciones en cuDF
month_trajectories = calcular_trayectorias(df_merged_dynamic)
month_stops = detectar_segmentos_parada_cudf(month_trajectories)
month_separated_trajectories = dividir_trayectorias_por_paradas(month_trajectories)

# Guardar resultados
month_trajectories.to_csv('trajectories_detected.csv', index=False)
month_stops.to_csv('stop_segments_detected.csv', index=False)
month_separated_trajectories.to_csv('trajectories_split_detected.csv', index=False)

```

MEJORA DE LA EFICIENCIA COMPUTACIONAL DE UN SISTEMA DE PREDICCIÓN DE TIPO DE BUQUE UTILIZANDO PARÁMETROS DE TRAYECTORIAS Y TÉCNICAS DE IA

```
Out[21]:
```

	mmsi	timestamp	lat	lon	speed	distance	time_diff	split_id
2186	232018697	1675204884	24.987058	-17.793093	11.0	14862.675550	1288.0	8
2187	232018697	1675205333	25.013247	-17.780293	11.0	3184.993348	449.0	8
1658	232018697	1675205790	25.046707	-17.764345	10.8	4052.718494	457.0	8
1659	232018697	1675208152	25.152068	-17.711468	10.7	12868.773282	2362.0	8
1660	232018697	1675209285	25.125102	-17.725257	10.7	3304.171975	1133.0	8

```
In [35]: ## Primera iteracion ##
```

```
In [36]: def clean(df):
    df = df.sort_values(by='timestamp') # Ordenar por timestamp

    if 'shiptype_x' in df.columns and 'shiptype_y' in df.columns:
        df = df.rename(columns={'shiptype_x': 'shiptype'}).drop(columns=['shiptype_y'])

    df = df.dropna()
    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), None) # Limitar Latitudes
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), None) # Limitar Longitudes

    return df
```

```
In [41]: def calcular_parametros_trayectoria(month_separated_trajectories):
    # Definir columnas del DataFrame de salida
    traj_parameters = ['split_id', 'traj_dist',
                      'traj_time', 'traj_mean_speed', 'traj_max_speed',
                      'traj_mean_acc', 'traj_mean_dec']

    # Agregar columnas de rangos de velocidad
    speed_ranges_list = [
        [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20], [0, 5, 10, 15, 20, 300]
    ]

    for sr_index, speed_ranges in enumerate(speed_ranges_list, start=0):
        for i in range(len(speed_ranges) - 1):
            col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
            traj_parameters.append(col_name)

    df_traj_parameters = cudf.DataFrame(columns=traj_parameters) # Crear un DataFrame vacío en cuDF
    grouped = month_separated_trajectories.groupby('split_id') # Agrupar por trayectorias
    df_traj_parameters['split_id'] = grouped['split_id'].first()

    # Calcular distancia total y tiempo entre mensajes en segundos
    df_traj_parameters['traj_dist'] = grouped['distance'].sum()
    df_traj_parameters['traj_time'] = grouped['timestamp'].diff().fillna(0) / 1e9
    # Cálculo de velocidades
    df_traj_parameters['traj_mean_speed'] = grouped['speed'].mean()
    df_traj_parameters['traj_max_speed'] = grouped['speed'].max()
    # Para cada conjunto de rangos de velocidad, crear una columna con el porcentaje de datos que caen en ese tramo
    for sr_index, speed_ranges in enumerate(speed_ranges_list, start=0):
        for i in range(len(speed_ranges) - 1):
            col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
            df_traj_parameters[col_name] = grouped.apply(
                lambda g: (
                    ((g['speed'] >= speed_ranges[i]) & (g['speed'] < speed_ranges[i+1])).sum()
                ) / len(g)
            ) # La variable g, hace referencia a subconjunto del DataFrame que comparte el mismo valor de 'split_id'
    df_traj_parameters['traj_mean_acc'] = grouped['speed'].diff().mean() # Calcular aceleraciones y deceleraciones en GPU
    df_traj_parameters['traj_mean_dec'] = grouped['speed'].diff().min()

    # Reemplazar infinitos por NaN primero
    df_traj_parameters = df_traj_parameters.replace([np.inf, -np.inf], np.nan)

    # Eliminar todas las filas que contengan al menos un NaN (incluidos los antiguos inf)
    df_traj_parameters = df_traj_parameters.dropna()

    return df_traj_parameters
```

```
In [42]: df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)
print(df_traj_parameters)
```

```
/tmp/ipykernel_358188/2678280612.py:31: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated,
and in a future version of pandas the grouping columns will be excluded from the operation. Either pass 'include_groups=False' to exclude the grou
pings or explicitly select the grouping columns after groupby to silence this warning.
df_traj_parameters[col_name] = grouped.apply(
```

```

split_id    traj_dist    traj_time    traj_mean_speed    traj_max_speed \
0      0.0    1.038451e+07    2.746000e-06    12.617153    15.9
1      1.0    6.512390e+01    3.240000e-07    0.150000    0.3
2      2.0    2.399159e+02    6.070000e-07    0.100000    0.2
3      3.0    8.943416e+01    7.660000e-07    0.150000    0.2
4      4.0    8.480971e+03    8.200000e-07    0.915000    5.8
5      5.0    5.389416e+00    3.770000e-07    0.000000    0.0
6      6.0    6.117146e+01    4.520000e-07    0.000000    0.0
7      7.0    6.506037e+00    3.510000e-07    0.000000    0.0
8      8.0    7.414310e+06    3.780000e-07    10.614512    11.8

traj_mean_acc    traj_mean_dec    traj_percent_0-0.5    traj_percent_0.5-1 \
0      0.00145    -5.6    0.002737    0.000912
1      0.00145    -5.6    1.000000    0.000000
2      0.00145    -5.6    1.000000    0.000000
3      0.00145    -5.6    1.000000    0.000000
4      0.00145    -5.6    0.800000    0.000000
5      0.00145    -5.6    1.000000    0.000000
6      0.00145    -5.6    1.000000    0.000000
7      0.00145    -5.6    1.000000    0.000000
8      0.00145    -5.6    0.001759    0.000000

traj_percent_1-1.5    ...    traj_percent_10-12    traj_percent_12-14 \
0      0.0    ...    0.052007    0.922445
1      0.0    ...    0.000000    0.000000
2      0.0    ...    0.000000    0.000000
3      0.0    ...    0.000000    0.000000
4      0.0    ...    0.000000    0.000000
5      0.0    ...    0.000000    0.000000
6      0.0    ...    0.000000    0.000000
7      0.0    ...    0.000000    0.000000
8      0.0    ...    0.887423    0.000000

traj_percent_14-16    traj_percent_16-18    traj_percent_18-20 \
0      0.008212    0.0    0.0
1      0.000000    0.0    0.0
2      0.000000    0.0    0.0
3      0.000000    0.0    0.0
4      0.000000    0.0    0.0
5      0.000000    0.0    0.0
6      0.000000    0.0    0.0
7      0.000000    0.0    0.0
8      0.000000    0.0    0.0

traj_percent_0-5    traj_percent_5-10    traj_percent_10-15 \
0      0.004562    0.012774    0.978102
1      1.000000    0.000000    0.000000
2      1.000000    0.000000    0.000000
3      1.000000    0.000000    0.000000
4      0.900000    0.100000    0.000000
5      1.000000    0.000000    0.000000
6      1.000000    0.000000    0.000000
7      1.000000    0.000000    0.000000
8      0.003518    0.109059    0.887423

traj_percent_15-20    traj_percent_20-300
0      0.004562    0.0
1      0.000000    0.0
2      0.000000    0.0
3      0.000000    0.0
4      0.000000    0.0
5      0.000000    0.0
6      0.000000    0.0
7      0.000000    0.0
8      0.000000    0.0

[9 rows x 42 columns]

```

```

In [51]: def calcular_parametros_paradas(month_stops):
          if month_stops.shape[0] == 0:
              # Crear un DataFrame con una fila llena de ceros
              df_empty = cudf.DataFrame({
                  'mmsi': [0],
                  'stop_max_lat': [0], 'stop_min_lat': [0], 'stop_mean_lat': [0],
                  'stop_max_lon': [0], 'stop_min_lon': [0], 'stop_mean_lon': [0],
                  'N_stops': [0],
                  'stop_time_max': [0], 'stop_time_min': [0], 'stop_time_mean': [0],
                  'stop_dist_max': [0], 'stop_dist_min': [0], 'stop_dist_mean': [0]
              })
          return df_empty # Retornar DataFrame con ceros
          # Agrupar por MMSI y calcular valores agregados
          grouped = month_stops.groupby('mmsi').agg({
              'lat': ['max', 'min', 'mean'],
              'lon': ['max', 'min', 'mean'],
              'split_id': 'count',
              'time_diff': ['max', 'min', 'mean']
          }).reset_index()
          # Ajustar nombres de columnas
          grouped.columns = ['mmsi', 'stop_max_lat', 'stop_min_lat', 'stop_mean_lat',
                              'stop_max_lon', 'stop_min_lon', 'stop_mean_lon',
                              'N_stops', 'stop_time_max', 'stop_time_min', 'stop_time_mean']
          # Cálculo de distancias de paradas sin usar apply_grouped
          stop_distances = month_stops.copy()
          stop_distances['stop_dist'] = (stop_distances.groupby(['mmsi', 'split_id'])['lat']
                                          .diff().abs() +
                                          stop_distances.groupby(['mmsi', 'split_id'])['lon']
                                          .diff().abs()).fillna(0)
          # Obtener las estadísticas de distancia (máxima, mínima y media)
          stop_dist_stats = stop_distances.groupby('mmsi').agg({
              'stop_dist': ['max', 'min', 'mean']
          })

```

MEJORA DE LA EFICIENCIA COMPUTACIONAL DE UN SISTEMA DE PREDICCIÓN DE TIPO DE BUQUE UTILIZANDO PARÁMETROS DE TRAYECTORIAS Y TÉCNICAS DE IA

```
)).reset_index()

stop_dist_stats.columns = ['mmsi', 'stop_dist_max', 'stop_dist_min', 'stop_dist_mean'] # Renombrar columnas correctamente
grouped = grouped.merge(stop_dist_stats, on='mmsi', how='left') # Fusionar resultados finales
return grouped.fillna(0)
```

In [52]:

```
df_month_stops = calcular_parametros_paradas(month_stops)

# Verificar si se han calculado correctamente los parámetros
if df_month_stops.shape[0] > 0:
    print("✅ Parámetros de paradas calculados correctamente.")
    print(df_month_stops.head())
else:
    print("⚠️ No se detectaron paradas correctamente. Verifica el DataFrame de entrada.")

✅ Parámetros de paradas calculados correctamente.
   mmsi  stop_max_lat  stop_min_lat  stop_mean_lat  stop_max_lon \
0 205059000      51.322843      51.322457      51.322594      3.202565

   stop_min_lon  stop_mean_lon  N_stops  stop_time_max  stop_time_min \
0      3.202198      3.202464       40      50440      3659

   stop_time_mean  stop_dist_max  stop_dist_min  stop_dist_mean
0      6978.45      0.000069      0.0      0.000005
```

In [53]:

```
def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories, df_month_stops):
    # Lista de parámetros a calcular
    parameters = ['mmsi', 'N_Trajectories',
                  'month_max_lat', 'month_min_lat', 'month_mean_lat',
                  'month_max_lon', 'month_min_lon', 'month_mean_lon',
                  'month_max_dif_lat', 'month_max_dif_lon',
                  'traj_dist_max', 'traj_dist_min', 'traj_dist_mean',
                  'month_dist', 'traj_time_mean',
                  'traj_mean_speed_mean', 'traj_max_speed_max', 'month_mean_speed',
                  'month_max_speed', 'traj_mean_acc_mean', 'traj_mean_dec_mean',
                  'N_stops', 'stop_dist_max', 'stop_dist_min', 'stop_dist_mean',
                  'stop_time_max', 'stop_time_min', 'stop_time_mean',
                  # Rangos de velocidad (Range 1)
                  'traj_percent_0_0dec5_mean', 'traj_percent_0dec5_1_mean',
                  'traj_percent_1_1dec5_mean', 'traj_percent_1dec5_2_mean',
                  'traj_percent_2_2dec5_mean', 'traj_percent_2dec5_3_mean',
                  'traj_percent_3_3dec5_mean', 'traj_percent_3dec5_4_mean',
                  'traj_percent_4_4dec5_mean', 'traj_percent_4dec5_5_mean',
                  # Rangos de velocidad (Range 2)
                  'traj_percent_0_1_mean', 'traj_percent_1_2_mean', 'traj_percent_2_3_mean',
                  'traj_percent_3_4_mean', 'traj_percent_4_5_mean', 'traj_percent_5_6_mean',
                  'traj_percent_6_7_mean', 'traj_percent_7_8_mean', 'traj_percent_8_9_mean',
                  'traj_percent_9_10_mean',
                  # Rangos de velocidad (Range 3)
                  'traj_percent_0_2_mean', 'traj_percent_2_4_mean', 'traj_percent_4_6_mean',
                  'traj_percent_6_8_mean', 'traj_percent_8_10_mean', 'traj_percent_10_12_mean',
                  'traj_percent_12_14_mean', 'traj_percent_14_16_mean', 'traj_percent_16_18_mean',
                  'traj_percent_18_20_mean',
                  # Rango de velocidad (Range 4)
                  'traj_percent_0_5_mean', 'traj_percent_5_10_mean', 'traj_percent_10_15_mean',
                  'traj_percent_15_20_mean', 'traj_percent_over20_mean'
                  ]

    # Crear un DataFrame vacío optimizado para GPU
    df_parameters_temp = cudf.DataFrame(columns=parameters)

    # Parámetros de paradas
    N_stops = df_month_stops['N_stops'].max()
    stop_dist_max = df_month_stops['stop_dist_max'].max()
    stop_dist_min = df_month_stops['stop_dist_min'].min()
    stop_dist_mean = df_month_stops['stop_dist_mean'].mean()

    stop_time_max = df_month_stops['stop_time_max'].max()
    stop_time_min = df_month_stops['stop_time_min'].min()
    stop_time_mean = df_month_stops['stop_time_mean'].mean()

    N_Trajectories = df_traj_parameters['split_id'].max()
    mmsi = month_trajectories['mmsi'].iloc[0]

    # Parámetros geográficos
    month_max_lat = month_trajectories['lat'].max()
    month_min_lat = month_trajectories['lat'].min()
    month_mean_lat = month_trajectories['lat'].mean()

    month_max_lon = month_trajectories['lon'].max()
    month_min_lon = month_trajectories['lon'].min()
    month_mean_lon = month_trajectories['lon'].mean()

    month_max_dif_lat = month_max_lat - month_min_lat
    month_max_dif_lon = month_max_lon - month_min_lon

    # Distancias
    month_dist = len(month_trajectories)
    traj_dist_max = df_traj_parameters['traj_dist'].max()
    traj_dist_min = df_traj_parameters['traj_dist'].min()
    traj_dist_mean = df_traj_parameters['traj_dist'].mean()

    # Aspectos Temporales
    traj_time_mean = df_traj_parameters['traj_time'].mean()

    # Velocidades
    traj_mean_speed_mean = df_traj_parameters['traj_mean_speed'].mean()
    month_mean_speed = month_trajectories['speed'].mean()
    traj_max_speed_max = df_traj_parameters['traj_max_speed'].max()
    month_max_speed = month_trajectories['speed'].max()
```

```

# Rango de velocidades 1
traj_percent_0_0dec5_mean = df_traj_parameters['traj_percent_0-0.5'].mean()
traj_percent_0dec5_1_mean = df_traj_parameters['traj_percent_0.5-1'].mean()
traj_percent_1_1dec5_mean = df_traj_parameters['traj_percent_1-1.5'].mean()
traj_percent_1dec5_2_mean = df_traj_parameters['traj_percent_1.5-2'].mean()
traj_percent_2_2dec5_mean = df_traj_parameters['traj_percent_2-2.5'].mean()
traj_percent_2dec5_3_mean = df_traj_parameters['traj_percent_2.5-3'].mean()
traj_percent_3_3dec5_mean = df_traj_parameters['traj_percent_3-3.5'].mean()
traj_percent_3dec5_4_mean = df_traj_parameters['traj_percent_3.5-4'].mean()
traj_percent_4_4dec5_mean = df_traj_parameters['traj_percent_4-4.5'].mean()
traj_percent_4dec5_5_mean = df_traj_parameters['traj_percent_4.5-5'].mean()

# Rango de velocidades 2
traj_percent_0_1_mean = df_traj_parameters['traj_percent_0-1'].mean()
traj_percent_1_2_mean = df_traj_parameters['traj_percent_1-2'].mean()
traj_percent_2_3_mean = df_traj_parameters['traj_percent_2-3'].mean()
traj_percent_3_4_mean = df_traj_parameters['traj_percent_3-4'].mean()
traj_percent_4_5_mean = df_traj_parameters['traj_percent_4-5'].mean()
traj_percent_5_6_mean = df_traj_parameters['traj_percent_5-6'].mean()
traj_percent_6_7_mean = df_traj_parameters['traj_percent_6-7'].mean()
traj_percent_7_8_mean = df_traj_parameters['traj_percent_7-8'].mean()
traj_percent_8_9_mean = df_traj_parameters['traj_percent_8-9'].mean()
traj_percent_9_10_mean = df_traj_parameters['traj_percent_9-10'].mean()

# Rango de velocidades 3
traj_percent_0_2_mean = df_traj_parameters['traj_percent_0-2'].mean()
traj_percent_2_4_mean = df_traj_parameters['traj_percent_2-4'].mean()
traj_percent_4_6_mean = df_traj_parameters['traj_percent_4-6'].mean()
traj_percent_6_8_mean = df_traj_parameters['traj_percent_6-8'].mean()
traj_percent_8_10_mean = df_traj_parameters['traj_percent_8-10'].mean()
traj_percent_10_12_mean = df_traj_parameters['traj_percent_10-12'].mean()
traj_percent_12_14_mean = df_traj_parameters['traj_percent_12-14'].mean()
traj_percent_14_16_mean = df_traj_parameters['traj_percent_14-16'].mean()
traj_percent_16_18_mean = df_traj_parameters['traj_percent_16-18'].mean()
traj_percent_18_20_mean = df_traj_parameters['traj_percent_18-20'].mean()

# Rango de velocidades 4
traj_percent_0_5_mean = df_traj_parameters['traj_percent_0-5'].mean()
traj_percent_5_10_mean = df_traj_parameters['traj_percent_5-10'].mean()
traj_percent_10_15_mean = df_traj_parameters['traj_percent_10-15'].mean()
traj_percent_15_20_mean = df_traj_parameters['traj_percent_15-20'].mean()
traj_percent_over20_mean = df_traj_parameters['traj_percent_20-300'].mean()

# Aceleraciones y deceleraciones
traj_mean_acc_mean = df_traj_parameters['traj_mean_acc'].mean()
traj_mean_dec_mean = df_traj_parameters['traj_mean_dec'].mean()

# Agregar Los resultados al DataFrame
df_parameters_temp = cudf.DataFrame({
    'mmsi': [mmsi],
    'N_Trajectories': [N_Trajectories],
    'month_max_lat': [month_max_lat],
    'month_min_lat': [month_min_lat],
    'month_mean_lat': [month_mean_lat],
    'month_max_lon': [month_max_lon],
    'month_min_lon': [month_min_lon],
    'month_mean_lon': [month_mean_lon],
    'month_max_diff_lat': [month_max_diff_lat],
    'month_max_diff_lon': [month_max_diff_lon],
    'traj_dist_max': [traj_dist_max],
    'traj_dist_min': [traj_dist_min],
    'traj_dist_mean': [traj_dist_mean],
    'month_dist': [month_dist],
    'traj_time_mean': [traj_time_mean],
    'traj_mean_speed_mean': [traj_mean_speed_mean],
    'traj_max_speed_max': [traj_max_speed_max],
    'month_mean_speed': [month_mean_speed],
    'month_max_speed': [month_max_speed],
    'traj_mean_acc_mean': [traj_mean_acc_mean],
    'traj_mean_dec_mean': [traj_mean_dec_mean],
    'N_stops': [N_stops],
    'stop_dist_max': [stop_dist_max],
    'stop_dist_min': [stop_dist_min],
    'stop_dist_mean': [stop_dist_mean],
    'stop_time_max': [stop_time_max],
    'stop_time_min': [stop_time_min],
    'stop_time_mean': [stop_time_mean],
    # Range 1
    'traj_percent_0_0dec5_mean': [traj_percent_0_0dec5_mean], 'traj_percent_0dec5_1_mean': [traj_percent_0dec5_1_mean],
    'traj_percent_1_1dec5_mean': [traj_percent_1_1dec5_mean], 'traj_percent_1dec5_2_mean': [traj_percent_1dec5_2_mean],
    'traj_percent_2_2dec5_mean': [traj_percent_2_2dec5_mean], 'traj_percent_2dec5_3_mean': [traj_percent_2dec5_3_mean],
    'traj_percent_3_3dec5_mean': [traj_percent_3_3dec5_mean], 'traj_percent_3dec5_4_mean': [traj_percent_3dec5_4_mean],
    'traj_percent_4_4dec5_mean': [traj_percent_4_4dec5_mean], 'traj_percent_4dec5_5_mean': [traj_percent_4dec5_5_mean],
    # Range 2
    'traj_percent_0_1_mean': [traj_percent_0_1_mean],
    'traj_percent_1_2_mean': [traj_percent_1_2_mean],
    'traj_percent_2_3_mean': [traj_percent_2_3_mean],
    'traj_percent_3_4_mean': [traj_percent_3_4_mean],
    'traj_percent_4_5_mean': [traj_percent_4_5_mean],
    'traj_percent_5_6_mean': [traj_percent_5_6_mean],
    'traj_percent_6_7_mean': [traj_percent_6_7_mean],
    'traj_percent_7_8_mean': [traj_percent_7_8_mean],
    'traj_percent_8_9_mean': [traj_percent_8_9_mean],
    'traj_percent_9_10_mean': [traj_percent_9_10_mean],
    # Range 3
    'traj_percent_0_2_mean': [traj_percent_0_2_mean], 'traj_percent_2_4_mean': [traj_percent_2_4_mean],
    'traj_percent_4_6_mean': [traj_percent_4_6_mean], 'traj_percent_6_8_mean': [traj_percent_6_8_mean],
    'traj_percent_8_10_mean': [traj_percent_8_10_mean], 'traj_percent_10_12_mean': [traj_percent_10_12_mean],
    'traj_percent_12_14_mean': [traj_percent_12_14_mean], 'traj_percent_14_16_mean': [traj_percent_14_16_mean],
    'traj_percent_16_18_mean': [traj_percent_16_18_mean], 'traj_percent_18_20_mean': [traj_percent_18_20_mean],
    # Range 4
    'traj_percent_0_5_mean': [traj_percent_0_5_mean], 'traj_percent_5_10_mean': [traj_percent_5_10_mean],

```

```

        'traj_percent_10_15_mean': [traj_percent_10_15_mean], 'traj_percent_15_20_mean': [traj_percent_15_20_mean],
        'traj_percent_over20_mean': [traj_percent_over20_mean]
    })

    return df_parameters_temp

```

In [54]: `df_cinematic_parameters_temp = calcular_parametros_cinematicos(df_traj_parameters, month_trajectories, df_month_stops)`
`df_cinematic_parameters_temp.head()`

Out[54]:

	mmsi	N_Trajectories	month_max_lat	month_min_lat	month_mean_lat	month_max_lon	month_min_lon	month_mean_lon	month_max_diff_lat	month_max_diff_lon
0	205059000	13.0	51.368288	51.093468	51.321776	3.222338	1.970073	3.197364	0.27482	0.27482

1 rows × 11 columns

In [47]: `df_cinematic_parameters_temp.isna().any().any()`

Out[47]: `np.False_`

In [38]: `df_cinematic_parameters_temp.to_csv('PRUEBA.csv', index=False)`

In [29]: `df_traj_parameters`

Out[29]:

	split_id	traj_dist	traj_time	traj_mean_speed	traj_max_speed	traj_mean_acc	traj_mean_dec	traj_percent_0-0.5	traj_percent_0.5-1	traj_percent_1-1.5	traj_percent_1.5-2	traj_percent_2-2.5	traj_percent_2.5-3	traj_percent_3-3.5	traj_percent_3.5-4	traj_percent_4-4.5	traj_percent_4.5-5	traj_percent_5-5.5	traj_percent_5.5-6	traj_percent_6-6.5	traj_percent_6.5-7	traj_percent_7-7.5	traj_percent_7.5-8	traj_percent_8-8.5	traj_percent_8.5-9	traj_percent_9-9.5	traj_percent_9.5-10	traj_percent_10-10.5	traj_percent_10.5-11	traj_percent_11-11.5	traj_percent_11.5-12	traj_percent_12-12.5	traj_percent_12.5-13	traj_percent_13-13.5	traj_percent_13.5-14	traj_percent_14-14.5	traj_percent_14.5-15	traj_percent_15-15.5	traj_percent_15.5-16	traj_percent_16-16.5	traj_percent_16.5-17	traj_percent_17-17.5	traj_percent_17.5-18	traj_percent_18-18.5	traj_percent_18.5-19	traj_percent_19-19.5	traj_percent_19.5-20	traj_percent_20-20.5	traj_percent_20.5-21	traj_percent_21-21.5	traj_percent_21.5-22	traj_percent_22-22.5	traj_percent_22.5-23	traj_percent_23-23.5	traj_percent_23.5-24	traj_percent_24-24.5	traj_percent_24.5-25	traj_percent_25-25.5	traj_percent_25.5-26	traj_percent_26-26.5	traj_percent_26.5-27	traj_percent_27-27.5	traj_percent_27.5-28	traj_percent_28-28.5	traj_percent_28.5-29	traj_percent_29-29.5	traj_percent_29.5-30	traj_percent_30-30.5	traj_percent_30.5-31	traj_percent_31-31.5	traj_percent_31.5-32	traj_percent_32-32.5	traj_percent_32.5-33	traj_percent_33-33.5	traj_percent_33.5-34	traj_percent_34-34.5	traj_percent_34.5-35	traj_percent_35-35.5	traj_percent_35.5-36	traj_percent_36-36.5	traj_percent_36.5-37	traj_percent_37-37.5	traj_percent_37.5-38	traj_percent_38-38.5	traj_percent_38.5-39	traj_percent_39-39.5	traj_percent_39.5-40	traj_percent_40-40.5	traj_percent_40.5-41	traj_percent_41-41.5	traj_percent_41.5-42	traj_percent_42-42.5	traj_percent_42.5-43	traj_percent_43-43.5	traj_percent_43.5-44	traj_percent_44-44.5	traj_percent_44.5-45	traj_percent_45-45.5	traj_percent_45.5-46	traj_percent_46-46.5	traj_percent_46.5-47	traj_percent_47-47.5	traj_percent_47.5-48	traj_percent_48-48.5	traj_percent_48.5-49	traj_percent_49-49.5	traj_percent_49.5-50	traj_percent_50-50.5	traj_percent_50.5-51	traj_percent_51-51.5	traj_percent_51.5-52	traj_percent_52-52.5	traj_percent_52.5-53	traj_percent_53-53.5	traj_percent_53.5-54	traj_percent_54-54.5	traj_percent_54.5-55	traj_percent_55-55.5	traj_percent_55.5-56	traj_percent_56-56.5	traj_percent_56.5-57	traj_percent_57-57.5	traj_percent_57.5-58	traj_percent_58-58.5	traj_percent_58.5-59	traj_percent_59-59.5	traj_percent_59.5-60	traj_percent_60-60.5	traj_percent_60.5-61	traj_percent_61-61.5	traj_percent_61.5-62	traj_percent_62-62.5	traj_percent_62.5-63	traj_percent_63-63.5	traj_percent_63.5-64	traj_percent_64-64.5	traj_percent_64.5-65	traj_percent_65-65.5	traj_percent_65.5-66	traj_percent_66-66.5	traj_percent_66.5-67	traj_percent_67-67.5	traj_percent_67.5-68	traj_percent_68-68.5	traj_percent_68.5-69	traj_percent_69-69.5	traj_percent_69.5-70	traj_percent_70-70.5	traj_percent_70.5-71	traj_percent_71-71.5	traj_percent_71.5-72	traj_percent_72-72.5	traj_percent_72.5-73	traj_percent_73-73.5	traj_percent_73.5-74	traj_percent_74-74.5	traj_percent_74.5-75	traj_percent_75-75.5	traj_percent_75.5-76	traj_percent_76-76.5	traj_percent_76.5-77	traj_percent_77-77.5	traj_percent_77.5-78	traj_percent_78-78.5	traj_percent_78.5-79	traj_percent_79-79.5	traj_percent_79.5-80	traj_percent_80-80.5	traj_percent_80.5-81	traj_percent_81-81.5	traj_percent_81.5-82	traj_percent_82-82.5	traj_percent_82.5-83	traj_percent_83-83.5	traj_percent_83.5-84	traj_percent_84-84.5	traj_percent_84.5-85	traj_percent_85-85.5	traj_percent_85.5-86	traj_percent_86-86.5	traj_percent_86.5-87	traj_percent_87-87.5	traj_percent_87.5-88	traj_percent_88-88.5	traj_percent_88.5-89	traj_percent_89-89.5	traj_percent_89.5-90	traj_percent_90-90.5	traj_percent_90.5-91	traj_percent_91-91.5	traj_percent_91.5-92	traj_percent_92-92.5	traj_percent_92.5-93	traj_percent_93-93.5	traj_percent_93.5-94	traj_percent_94-94.5	traj_percent_94.5-95	traj_percent_95-95.5	traj_percent_95.5-96	traj_percent_96-96.5	traj_percent_96.5-97	traj_percent_97-97.5	traj_percent_97.5-98	traj_percent_98-98.5	traj_percent_98.5-99	traj_percent_99-99.5	traj_percent_99.5-100
0	0	1.038449e+07	2746.0	12.640037	15.9	0.00158	-5.6	0.000914	0.000914	0.0	...																																																																																																																																																																																																				
1	1	2.349560e+02	324.0	0.137500	0.3	0.00158	-5.6	1.000000	0.000000	0.0	...																																																																																																																																																																																																				
2	2	1.822234e+02	607.0	0.100000	0.2	0.00158	-5.6	1.000000	0.000000	0.0	...																																																																																																																																																																																																				
3	3	1.683416e+02	766.0	0.100000	0.1	0.00158	-5.6	1.000000	0.000000	0.0	...																																																																																																																																																																																																				
4	4	5.890501e+01	820.0	0.150000	0.2	0.00158	-5.6	1.000000	0.000000	0.0	...																																																																																																																																																																																																				
5	5	8.241498e+03	377.0	2.471429	5.8	0.00158	-5.6	0.428571	0.000000	0.0	...																																																																																																																																																																																																				
6	6	7.414396e+06	452.0	10.413028	11.8	0.00158	-5.6	0.020708	0.000000	0.0	...																																																																																																																																																																																																				

7 rows × 13 columns

In [48]: `print(df_merged_dynamic.isnull().any().any())`
`print(month_trajectories.isnull().any().any())`
`print(df_traj_parameters.isnull().any().any())`
`print(df_month_stops.isnull().any().any())`
`print(df_cinematic_parameters_temp.isnull().any().any())`

False
False
False
False
False

In [22]: `import warnings`
`# Desactivar RuntimeWarnings de cudf`
`warnings.filterwarnings('ignore', category=RuntimeWarning)`

In [55]: `# Leer el DataFrame que contiene La Lista de MMSI y SHIPTYPE`
`df_MMSI_shiptype = cudf.read_csv('MMSI_shiptype_data.csv')`
`df_MMSI_shiptype['MMSI'] = df_MMSI_shiptype['MMSI'].astype('int64')`
`df_MMSI_shiptype['shiptype'] = df_MMSI_shiptype['shiptype'].astype(str).str.strip().str.lower()`
`df_MMSI_shiptype = df_MMSI_shiptype.rename(columns={'MMSI': 'mmsi'})`
`total_time = 0`
`containing_nan = 0`
`too_short_csv = 0`
`non_associated_shiptypes = 0`
`tiempos_procesamiento = cudf.DataFrame(columns=['mmsi', 'tiempo_procesamiento'])`
`df_cinematic_parameters = df_MMSI_shiptype.copy()`
`df_cinematic_results = cudf.DataFrame()`
`for index in tqdm(range(len(df_cinematic_parameters)), desc="Processing MMSIs"):`
`row = df_cinematic_parameters.iloc[index]`
`start_time = time.time()`
`csv_file = os.path.join('Dynamic_by_MMSI_month', f"{row['mmsi']}[index].csv")`
`if os.path.exists(csv_file):`
`df_raw_dynamic = cudf.read_csv(csv_file)`
`df_raw_dynamic['mmsi'] = df_raw_dynamic['mmsi'].astype('int64')`
`if len(df_raw_dynamic) >= 100:`
`df_merged_dynamic = df_raw_dynamic.merge(df_MMSI_shiptype, on='mmsi', how='inner')`
`if df_merged_dynamic.empty:`
`non_associated_shiptypes += 1`
`continue`
`df_merged_dynamic = clean(df_merged_dynamic)`
`df_merged_dynamic = df_merged_dynamic.dropna()`
`if not df_merged_dynamic.isnull().any().any():`
`# Cálculo de distancia`

```

df_merged_dynamic['lat_shift'] = df_merged_dynamic['lat'].shift()
df_merged_dynamic['lon_shift'] = df_merged_dynamic['lon'].shift()
# Calcular el tiempo entre puntos (en segundos)
df_merged_dynamic['time_diff'] = df_merged_dynamic.groupby('mmsi')['timestamp'].diff().astype('int64')

try:
    df_merged_dynamic.drop(index=df_merged_dynamic.index[0], axis=0, inplace=True)
except IndexError as e:
    print(f"⚠ Se detectó un IndexError en la iteración {index}, omitiendo...")
    continue # Evita que el bucle se detenga

# Asignación de valores para la fórmula de Haversine
lat1 = cp.asarray(df_merged_dynamic['lat'].values)
lon1 = cp.asarray(df_merged_dynamic['lon'].values)
lat2 = cp.asarray(df_merged_dynamic['lat_shift'].values)
lon2 = cp.asarray(df_merged_dynamic['lon_shift'].values)
df_merged_dynamic['distance'] = haversine_gpu(lat1, lon1, lat2, lon2).get()

# Eliminar valores nulos
df_merged_dynamic = df_merged_dynamic.dropna()
df_merged_dynamic = clean(df_merged_dynamic)

# Obtener las trayectorias creadas previamente
month_trajectories = cudf.DataFrame()
month_trajectories = calcular_trayectorias(df_merged_dynamic)

# Separación de trayectorias y detección de paradas
month_separated_trajectories = dividir_trayectorias_por_paradas(month_trajectories)
month_stops = detectar_segmentos_parada_cudf(month_trajectories)

# Calcular parámetros de trayectoria y paradas
df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)

if month_stops.shape[0] == 0:
    df_month_stops = calcular_parametros_paradas(month_stops)
else: df_month_stops = calcular_parametros_paradas(month_stops)

# Calcular parámetros cinemáticos
df_cinematic_parameters_temp = calcular_parametros_cinematicos(df_traj_parameters, month_trajectories, df_month_stops)
# Asegurar que todas las columnas coincidan
if df_cinematic_results.shape[0] == 0: # Si es la primera iteración, definir columnas
    df_cinematic_results = cudf.DataFrame(columns=df_cinematic_parameters_temp.columns)

# Asegurar que 'mmsi' esté en 'df_cinematic_parameters_temp'
df_cinematic_parameters_temp['mmsi'] = df_merged_dynamic['mmsi']

# asegurar que todas las columnas coincidan
df_cinematic_parameters_temp = df_cinematic_parameters_temp[df_cinematic_results.columns]

# Concatenar filas
df_cinematic_results = cudf.concat([df_cinematic_results, df_cinematic_parameters_temp], ignore_index=True)

if 'shiptype' in df_merged_dynamic.columns:
    shiptype = df_merged_dynamic['shiptype'].iloc[0]
else:
    shiptype = 'unknown'
else:
    containing_nan += 1
else:
    too_short_csv += 1
else:
    non_associated_shiptypes += 1

total_time += time.time() - start_time
tiempos_procesamiento = cudf.concat([tiempos_procesamiento, cudf.DataFrame({'mmsi': [row['mmsi']], 'tiempo_procesamiento': [time.time() - start_time]}), ignore_index=True])

# Fusionar los resultados con el DataFrame original basado en 'mmsi'
df_cinematic_parameters = df_cinematic_parameters.merge(df_cinematic_results, on='mmsi', how='left')

print("Existen", containing_nan, "MMSI cuyo df contiene NaN.")
print("Existen", too_short_csv, "MMSI que no cumplen con la longitud requerida.")
print("Existen", non_associated_shiptypes, "MMSI que no se han asociado a ningún shiptype.")
print("Tiempo total en extraer parámetros:", total_time, "segundos.")

Processing MMSIs: 58%|██████████| 23764/40859 [9:39:21<5:27:16, 1.15s/it]
⚠ Se detectó un IndexError en la iteración 23764, omitiendo...
Processing MMSIs: 100%|██████████| 40859/40859 [16:33:22<00:00, 1.46s/it]
Existen 0 MMSI cuyo df contiene NaN.
Existen 3719 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 59381.09842133522 segundos.

In [86]: df_cinematic_parameters = df_cinematic_parameters.dropna()

In [89]: df_cinematic_parameters.isnull().any().any()
print(df_cinematic_parameters.shape)

(36824, 64)

In [56]: df_cinematic_parameters.to_csv('cinematic_parameters_1_jan.csv', index=False)

In [ ]: # Segunda iteración --> SE ELIMINAN LOS PARAMETROS MENOS IMPORTANTES PARA RANDOM FOREST

In [42]: def clean(df):
    df = df.sort_values(by='timestamp') # Ordenar por timestamp

    if 'shiptype_x' in df.columns and 'shiptype_y' in df.columns:
        df = df.rename(columns={'shiptype_x': 'shiptype'}).drop(columns=['shiptype_y'])

```

```
# Crear un DataFrame vacío optimizado para GPU
df_parameters_temp = cudf.DataFrame(columns=parameters)

mmsi = month_trajectories['mmsi'].iloc[0]

# Parámetros geográficos
month_max_lat = month_trajectories['lat'].max()
month_min_lat = month_trajectories['lat'].min()
month_mean_lat = month_trajectories['lat'].mean()

month_max_lon = month_trajectories['lon'].max()
month_min_lon = month_trajectories['lon'].min()
month_mean_lon = month_trajectories['lon'].mean()

month_max_dif_lat = month_max_lat - month_min_lat
month_max_dif_lon = month_max_lon - month_min_lon

# Distancias
month_dist = len(month_trajectories)
traj_dist_max = df_traj_parameters['traj_dist'].max()
traj_dist_min = df_traj_parameters['traj_dist'].min()
traj_dist_mean = df_traj_parameters['traj_dist'].mean()

# Aspectos Temporales
traj_time_mean = df_traj_parameters['traj_time'].mean()

# Velocidades
traj_mean_speed_mean = df_traj_parameters['traj_mean_speed'].mean()
month_mean_speed = month_trajectories['speed'].mean()
traj_max_speed_max = df_traj_parameters['traj_max_speed'].max()
month_max_speed = month_trajectories['speed'].max()

# Rango de velocidades 1
traj_percent_0dec5_1_mean = df_traj_parameters['traj_percent_0.5-1'].mean()
traj_percent_2dec5_3_mean = df_traj_parameters['traj_percent_2.5-3'].mean()
traj_percent_4dec5_5_mean = df_traj_parameters['traj_percent_4.5-5'].mean()
# Rango de velocidades 2
traj_percent_1_2_mean = df_traj_parameters['traj_percent_1-2'].mean()
traj_percent_2_3_mean = df_traj_parameters['traj_percent_2-3'].mean()
traj_percent_3_4_mean = df_traj_parameters['traj_percent_3-4'].mean()
traj_percent_4_5_mean = df_traj_parameters['traj_percent_4-5'].mean()
traj_percent_5_6_mean = df_traj_parameters['traj_percent_5-6'].mean()
traj_percent_9_10_mean = df_traj_parameters['traj_percent_9-10'].mean()
# Rango de velocidades 3
traj_percent_2_4_mean = df_traj_parameters['traj_percent_2-4'].mean()
traj_percent_4_6_mean = df_traj_parameters['traj_percent_4-6'].mean()
traj_percent_8_10_mean = df_traj_parameters['traj_percent_8-10'].mean()
traj_percent_10_12_mean = df_traj_parameters['traj_percent_10-12'].mean()
traj_percent_12_14_mean = df_traj_parameters['traj_percent_12-14'].mean()
traj_percent_14_16_mean = df_traj_parameters['traj_percent_14-16'].mean()
# Rango de velocidades 4
traj_percent_10_15_mean = df_traj_parameters['traj_percent_10-15'].mean()
# Aceleraciones y deceleraciones
traj_mean_dec_mean = df_traj_parameters['traj_mean_dec'].mean()

# Agregar Los resultados al DataFrame
df_parameters_temp = cudf.DataFrame({
    'mmsi': [mmsi],
    'month_max_lat': [month_max_lat],
    'month_min_lat': [month_min_lat],
    'month_mean_lat': [month_mean_lat],
    'month_max_lon': [month_max_lon],
    'month_min_lon': [month_min_lon],
    'month_mean_lon': [month_mean_lon],
    'month_max_dif_lat': [month_max_dif_lat],
    'month_max_dif_lon': [month_max_dif_lon],
    'traj_dist_max': [traj_dist_max],
    'traj_dist_mean': [traj_dist_mean],
    'month_dist': [month_dist],
    'traj_time_mean': [traj_time_mean],
    'traj_mean_speed_mean': [traj_mean_speed_mean],
    'traj_max_speed_max': [traj_max_speed_max],
    'month_mean_speed': [month_mean_speed],
    'month_max_speed': [month_max_speed],
    'traj_mean_dec_mean': [traj_mean_dec_mean],
    # Range 1
    'traj_percent_0dec5_1_mean': [traj_percent_0dec5_1_mean],
    'traj_percent_2dec5_3_mean': [traj_percent_2dec5_3_mean],
    'traj_percent_4dec5_5_mean': [traj_percent_2dec5_3_mean],
    # Range 2
    'traj_percent_1_2_mean': [traj_percent_1_2_mean],
    'traj_percent_2_3_mean': [traj_percent_2_3_mean],
    'traj_percent_3_4_mean': [traj_percent_3_4_mean],
    'traj_percent_4_5_mean': [traj_percent_4_5_mean],
    'traj_percent_5_6_mean': [traj_percent_5_6_mean],
    'traj_percent_9_10_mean': [traj_percent_9_10_mean],
    # Range 3
    'traj_percent_2_4_mean': [traj_percent_2_4_mean],
    'traj_percent_4_6_mean': [traj_percent_4_6_mean],
    'traj_percent_8_10_mean': [traj_percent_8_10_mean],
    'traj_percent_10_12_mean': [traj_percent_10_12_mean],
    'traj_percent_12_14_mean': [traj_percent_12_14_mean],
    'traj_percent_14_16_mean': [traj_percent_14_16_mean],
    # Range 4
    'traj_percent_10_15_mean': [traj_percent_10_15_mean],
})

return df_parameters_temp
```

```

In [46]: # Leer el DataFrame que contiene la lista de MMSI y SHIPTYPE
df_MMSI_shiptype = cudf.read_csv('MMSI_shiptype_data.csv')
df_MMSI_shiptype['MMSI'] = df_MMSI_shiptype['MMSI'].astype('int64')
df_MMSI_shiptype['shiptype'] = df_MMSI_shiptype['shiptype'].astype(str).str.strip().str.lower()

df_MMSI_shiptype = df_MMSI_shiptype.rename(columns={'MMSI': 'mmsi'})

total_time = 0
containing_nan = 0
too_short_csv = 0
non_associated_shiptypes = 0
tiempos_procesamiento = cudf.DataFrame(columns=['mmsi', 'tiempo_procesamiento'])

df_cinematic_parameters = df_MMSI_shiptype.copy()
df_cinematic_results = cudf.DataFrame()

for index in tqdm(range(len(df_cinematic_parameters)), desc="Processing MMSIs"):
    row = df_cinematic_parameters.iloc[index]
    start_time = time.time()
    csv_file = os.path.join('Dynamic_by_MMSI_month', f"{row['mmsi']}[index].csv")

    if os.path.exists(csv_file):
        df_raw_dynamic = cudf.read_csv(csv_file)
        df_raw_dynamic['mmsi'] = df_raw_dynamic['mmsi'].astype('int64')

        if len(df_raw_dynamic) >= 100:
            df_merged_dynamic = df_raw_dynamic.merge(df_MMSI_shiptype, on='mmsi', how='inner')

            if df_merged_dynamic.empty:
                non_associated_shiptypes += 1
                continue

            df_merged_dynamic = clean(df_merged_dynamic)
            df_merged_dynamic = df_merged_dynamic.dropna()

            if not df_merged_dynamic.isnull().any().any():
                # Cálculo de distancia
                df_merged_dynamic['lat_shift'] = df_merged_dynamic['lat'].shift()
                df_merged_dynamic['lon_shift'] = df_merged_dynamic['lon'].shift()
                # Calcular el tiempo entre puntos (en segundos)
                df_merged_dynamic['time_diff'] = df_merged_dynamic.groupby('mmsi')['timestamp'].diff().astype('int64')

                try:
                    df_merged_dynamic.drop(index=df_merged_dynamic.index[0], axis=0, inplace=True)
                except IndexError as e:
                    print(f"⚠ Se detectó un IndexError en la iteración {index}, omitiendo...")
                    continue # Evita que el bucle se detenga

                # Asignación de valores para la fórmula de Haversine
                lat1 = cp.asarray(df_merged_dynamic['lat'].values)
                lon1 = cp.asarray(df_merged_dynamic['lon'].values)
                lat2 = cp.asarray(df_merged_dynamic['lat_shift'].values)
                lon2 = cp.asarray(df_merged_dynamic['lon_shift'].values)
                df_merged_dynamic['distance'] = haversine_gpu(lat1, lon1, lat2, lon2).get()

                # Eliminar valores nulos
                df_merged_dynamic = df_merged_dynamic.dropna()
                df_merged_dynamic = clean(df_merged_dynamic)

                # Obtener las trayectorias creadas previamente
                month_trajectories = cudf.DataFrame()
                month_trajectories = calcular_trayectorias(df_merged_dynamic)

                # Separación de trayectorias y detección de paradas
                month_separated_trajectories = dividir_trayectorias_por_paradas(month_trajectories)

                # Calcular parámetros de trayectoria y paradas
                df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)

                # Calcular parámetros cinemáticos
                df_cinematic_parameters_temp = calcular_parametros_cinematicos(df_traj_parameters, month_trajectories)
                # Asegurar que todas las columnas coinciden
                if df_cinematic_results.shape[0] == 0: # Si es la primera iteración, definir columnas
                    df_cinematic_results = cudf.DataFrame(columns=df_cinematic_parameters_temp.columns)

                # Asegurar que 'mmsi' está en 'df_cinematic_parameters_temp'
                df_cinematic_parameters_temp['mmsi'] = df_merged_dynamic['mmsi']

                # Segurar que todas las columnas coinciden
                df_cinematic_parameters_temp = df_cinematic_parameters_temp[df_cinematic_results.columns]

                # Concatenar filas
                df_cinematic_results = cudf.concat([df_cinematic_results, df_cinematic_parameters_temp], ignore_index=True)

                if 'shiptype' in df_merged_dynamic.columns:
                    shiptype = df_merged_dynamic['shiptype'].iloc[0]
                else:
                    shiptype = 'unknown'

            else:
                containing_nan += 1
        else:
            too_short_csv += 1
    else:
        non_associated_shiptypes += 1

    total_time += time.time() - start_time
    tiempos_procesamiento = cudf.concat([tiempos_procesamiento, cudf.DataFrame({'mmsi': [row['mmsi']], 'tiempo_procesamiento': [time.time() - sta

# Fusionar los resultados con el DataFrame original basado en 'mmsi'

```

```
df_cinematic_parameters = df_cinematic_parameters.merge(df_cinematic_results, on='mmsi', how='left')

print("Existen", containing_nan, "MMSI cuyo df contiene NaN.")
print("Existen", too_short_csv, "MMSI que no cumplen con la longitud requerida.")
print("Existen", non_associated_shiptypes, "MMSI que no se han asociado a ningún shiptype.")
print("Tiempo total en extraer parámetros:", total_time, "segundos.")

Processing MMSIs: 58%|██████████| 23764/40859 [4:47:29<2:41:50, 1.76it/s]
⚠ Se detectó un IndexError en la iteración 23764, omitiendo...
Processing MMSIs: 100%|██████████| 40859/40859 [7:57:25<00:00, 1.43it/s]
Existen 0 MMSI cuyo df contiene NaN.
Existen 3719 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 28409.429589509964 segundos.
```

```
In [47]: df_cinematic_parameters = df_cinematic_parameters.dropna()

In [48]: df_cinematic_parameters.to_csv('cinematic_parameters_2_jan_RF.csv', index=False)

In [ ]: # SEGUIMOS ELIMINANDO PARAMETROS

In [ ]: def calcular_parametros_trayectoria(month_separated_trajectories):
    # Definir columnas del DataFrame de salida
    traj_parameters = ['split_id', 'traj_dist',
                      'traj_time', 'traj_mean_speed', 'traj_max_speed',
                      'traj_mean_acc', 'traj_mean_dec']

    # Agregar columnas de rangos de velocidad
    speed_ranges_list = [
        [2, 4],
        [12, 14],
        [10, 15]
    ]

    for sr_index, speed_ranges in enumerate(speed_ranges_list, start=0):
        for i in range(len(speed_ranges) - 1):
            col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
            traj_parameters.append(col_name)

    # Crear un DataFrame vacío en cuDF
    df_traj_parameters = cudf.DataFrame(columns=traj_parameters)

    # Agrupar por trayectorias
    grouped = month_separated_trajectories.groupby('split_id')
    df_traj_parameters['split_id'] = grouped['split_id'].first()

    # Calcular distancia total y tiempo entre mensajes en segundos
    df_traj_parameters['traj_dist'] = grouped['distance'].sum()
    df_traj_parameters['traj_time'] = grouped['timestamp'].diff().fillna(0) / 1e9

    # Cálculo de velocidades
    df_traj_parameters['traj_mean_speed'] = grouped['speed'].mean()
    df_traj_parameters['traj_max_speed'] = grouped['speed'].max()

    # Para cada conjunto de rangos de velocidad, crear una columna con el porcentaje de datos que caen en ese tramo
    for sr_index, speed_ranges in enumerate(speed_ranges_list, start=0):
        for i in range(len(speed_ranges) - 1):
            col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
            df_traj_parameters[col_name] = grouped.apply(
                lambda g: (
                    ((g['speed'] >= speed_ranges[i]) & (g['speed'] < speed_ranges[i+1])).sum()
                ) / len(g)
            )

    # La variable g, hace referencia a subconjunto del DataFrame que comparte el mismo valor de 'split_id'
    # Calcular aceleraciones y deceleraciones en GPU
    df_traj_parameters['traj_mean_acc'] = grouped['speed'].diff().mean()
    df_traj_parameters['traj_mean_dec'] = grouped['speed'].diff().min()

    return df_traj_parameters

In [ ]: df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)
print(df_traj_parameters.columns)

In [ ]: df_traj_parameters

In [ ]: def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories):
    # Lista de parámetros a calcular
    parameters = ['mmsi',
                  'month_max_lat', 'month_min_lat', 'month_mean_lat',
                  'month_max_lon', 'month_min_lon', 'month_mean_lon',
                  'month_max_dif_lat', 'month_max_dif_lon',
                  'traj_dist_max', 'traj_dist_mean',
                  'month_dist', 'traj_time_mean',
                  'traj_mean_speed_mean', 'traj_max_speed_max', 'month_mean_speed',
                  'month_max_speed', 'traj_mean_dec_mean',
                  # Rangos de velocidad (Range 3)
                  'traj_percent_2_4_mean',
                  'traj_percent_12_14_mean',
                  # Rango de velocidad (Range 4)
                  'traj_percent_10_15_mean'
    ]

    # Crear un DataFrame vacío optimizado para GPU
    df_parameters_temp = cudf.DataFrame(columns=parameters)

    mmsi = month_trajectories['mmsi'].iloc[0]

    # Parámetros geográficos
```

```

month_max_lat = month_trajectories['lat'].max()
month_min_lat = month_trajectories['lat'].min()
month_mean_lat = month_trajectories['lat'].mean()

month_max_lon = month_trajectories['lon'].max()
month_min_lon = month_trajectories['lon'].min()
month_mean_lon = month_trajectories['lon'].mean()

month_max_dif_lat = month_max_lat - month_min_lat
month_max_dif_lon = month_max_lon - month_min_lon

# Distancias
month_dist = len(month_trajectories)
traj_dist_max = df_traj_parameters['traj_dist'].max()
traj_dist_min = df_traj_parameters['traj_dist'].min()
traj_dist_mean = df_traj_parameters['traj_dist'].mean()

# Aspectos Temporales
traj_time_mean = df_traj_parameters['traj_time'].mean()

# Velocidades
traj_mean_speed_mean = df_traj_parameters['traj_mean_speed'].mean()
month_mean_speed = month_trajectories['speed'].mean()
traj_max_speed_max = df_traj_parameters['traj_max_speed'].max()
month_max_speed = month_trajectories['speed'].max()

# Rango de velocidades 3
traj_percent_2_4_mean = df_traj_parameters['traj_percent_2-4'].mean()
traj_percent_12_14_mean = df_traj_parameters['traj_percent_12-14'].mean()
# Rango de velocidades 4
traj_percent_10_15_mean = df_traj_parameters['traj_percent_10-15'].mean()
# Aceleraciones y deceleraciones
traj_mean_dec_mean = df_traj_parameters['traj_mean_dec'].mean()

# Agregar Los resultados al DataFrame
df_parameters_temp = cudf.DataFrame({
    'mmsi': [mmsi],
    'month_max_lat': [month_max_lat],
    'month_min_lat': [month_min_lat],
    'month_mean_lat': [month_mean_lat],
    'month_max_lon': [month_max_lon],
    'month_min_lon': [month_min_lon],
    'month_mean_lon': [month_mean_lon],
    'month_max_dif_lat': [month_max_dif_lat],
    'month_max_dif_lon': [month_max_dif_lon],
    'traj_dist_max': [traj_dist_max],
    'traj_dist_mean': [traj_dist_mean],
    'month_dist': [month_dist],
    'traj_time_mean': [traj_time_mean],
    'traj_mean_speed_mean': [traj_mean_speed_mean],
    'traj_max_speed_max': [traj_max_speed_max],
    'month_mean_speed': [month_mean_speed],
    'month_max_speed': [month_max_speed],
    'traj_mean_dec_mean': [traj_mean_dec_mean],
    # Range 3
    'traj_percent_2_4_mean': [traj_percent_2_4_mean],
    'traj_percent_12_14_mean': [traj_percent_12_14_mean],
    # Range 4
    'traj_percent_10_15_mean': [traj_percent_10_15_mean],
})

return df_parameters_temp

```

```

In [34]: # Leer el DataFrame que contiene La Lista de MMSI y SHIPTYPE
df_MMSI_shiptype = cudf.read_csv('MMSI_shiptype_data.csv')
df_MMSI_shiptype['MMSI'] = df_MMSI_shiptype['MMSI'].astype('int64')
df_MMSI_shiptype['shiptype'] = df_MMSI_shiptype['shiptype'].astype(str).str.strip().str.lower()

df_MMSI_shiptype = df_MMSI_shiptype.rename(columns={'MMSI': 'mmsi'})

total_time = 0
containing_nan = 0
too_short_csv = 0
non_associated_shiptypes = 0
tiempos_procesamiento = cudf.DataFrame(columns=['mmsi', 'tiempo_procesamiento'])

df_cinematic_parameters = df_MMSI_shiptype.copy()
df_cinematic_results = cudf.DataFrame()

for index in tqdm(range(len(df_cinematic_parameters)), desc="Processing MMSIs"):
    row = df_cinematic_parameters.iloc[index]
    start_time = time.time()
    csv_file = os.path.join('Dynamic_by_MMSI_month', f"{row['mmsi']}[index]}.csv")

    if os.path.exists(csv_file):
        df_raw_dynamic = cudf.read_csv(csv_file)
        df_raw_dynamic['mmsi'] = df_raw_dynamic['mmsi'].astype('int64')

        if len(df_raw_dynamic) >= 100:
            df_merged_dynamic = df_raw_dynamic.merge(df_MMSI_shiptype, on='mmsi', how='inner')

            if df_merged_dynamic.empty:
                non_associated_shiptypes += 1
                continue

            df_merged_dynamic = clean(df_merged_dynamic)
            df_merged_dynamic = df_merged_dynamic.dropna()

```

MEJORA DE LA EFICIENCIA COMPUTACIONAL DE UN SISTEMA DE PREDICCIÓN DE TIPO DE BUQUE UTILIZANDO PARÁMETROS DE TRAYECTORIAS Y TÉCNICAS DE IA

```
if not df_merged_dynamic.isnull().any().any():
    # Cálculo de distancia
    df_merged_dynamic['lat_shift'] = df_merged_dynamic['lat'].shift()
    df_merged_dynamic['lon_shift'] = df_merged_dynamic['lon'].shift()
    # Calcular el tiempo entre puntos (en segundos)
    df_merged_dynamic['time_diff'] = df_merged_dynamic.groupby('mmsi')['timestamp'].diff().astype('int64')

    try:
        df_merged_dynamic.drop(index=df_merged_dynamic.index[0], axis=0, inplace=True)
    except IndexError as e:
        print(f"⚠ Se detectó un IndexError en la iteración {index}, omitiendo...")
        continue # Evita que el bucle se detenga

    # Asignación de valores para la fórmula de Haversine
    lat1 = cp.asarray(df_merged_dynamic['lat'].values)
    lon1 = cp.asarray(df_merged_dynamic['lon'].values)
    lat2 = cp.asarray(df_merged_dynamic['lat_shift'].values)
    lon2 = cp.asarray(df_merged_dynamic['lon_shift'].values)
    df_merged_dynamic['distance'] = haversine_gpu(lat1, lon1, lat2, lon2).get()

    # Eliminar valores nulos
    df_merged_dynamic = df_merged_dynamic.dropna()
    df_merged_dynamic = clean(df_merged_dynamic)

    # Obtener las trayectorias creadas previamente
    month_trajectories = cudf.DataFrame()
    month_trajectories = calcular_trayectorias(df_merged_dynamic)

    # Separación de trayectorias y detección de paradas
    month_separated_trajectories = dividir_trayectorias_por_paradas(month_trajectories)

    # Calcular parámetros de trayectoria y paradas
    df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)

    # Calcular parámetros cinemáticos
    df_cinematic_parameters_temp = calcular_parametros_cinematicos(df_traj_parameters, month_trajectories)
    # Asegurar que todas las columnas coinciden
    if df_cinematic_results.shape[0] == 0: # Si es la primera iteración, definir columnas
        df_cinematic_results = cudf.DataFrame(columns=df_cinematic_parameters_temp.columns)

    # Asegurar que 'mmsi' está en 'df_cinematic_parameters_temp'
    df_cinematic_parameters_temp['mmsi'] = df_merged_dynamic['mmsi']

    # asegurar que todas las columnas coinciden
    df_cinematic_parameters_temp = df_cinematic_parameters_temp[df_cinematic_results.columns]

    # Concatenar filas
    df_cinematic_results = cudf.concat([df_cinematic_results, df_cinematic_parameters_temp], ignore_index=True)

    if 'shiptype' in df_merged_dynamic.columns:
        shiptype = df_merged_dynamic['shiptype'].iloc[0]
    else:
        shiptype = 'unknown'

    else:
        containing_nan += 1
    else:
        too_short_csv += 1
    else:
        non_associated_shiptypes += 1

    total_time += time.time() - start_time
    tiempos_procesamiento = cudf.concat([tiempos_procesamiento, cudf.DataFrame({'mmsi': [row['mmsi']], 'tiempo_procesamiento': [time.time() - sta

# Fusionar los resultados con el DataFrame original basado en 'mmsi'
df_cinematic_parameters_3 = df_cinematic_parameters.merge(df_cinematic_results, on='mmsi', how='left')

print("Existen", containing_nan, "MMSI cuyo df contiene NaN.")
print("Existen", too_short_csv, "MMSI que no cumplen con la longitud requerida.")
print("Existen", non_associated_shiptypes, "MMSI que no se han asociado a ningún shiptype.")
print("Tiempo total en extraer parámetros:", total_time, "segundos.")

Processing MMSIs: 58%|██████████| 23764/40859 [1:32:25<46:40, 6.11it/s]
⚠ Se detectó un IndexError en la iteración 23764, omitiendo...
Processing MMSIs: 100%|██████████| 40859/40859 [2:29:23<00:00, 4.56it/s]
Existen 0 MMSI cuyo df contiene NaN.
Existen 3719 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 8727.39570403099 segundos.
```

```
In [36]: df_cinematic_parameters_3.to_csv('cinematic_parameters_3_jan_RF.csv', index=False)

In [ ]: # SEGUNDA ITERACION --> SE ELIMINAN LOS PARAMETROS MENOS IMPORTANTES PARA XGBOOST

In [37]: def clean(df):
    df = df.sort_values(by='timestamp') # Ordenar por timestamp

    if 'shiptype_x' in df.columns and 'shiptype_y' in df.columns:
        df = df.rename(columns={'shiptype_x': 'shiptype'}).drop(columns=['shiptype_y'])

    df = df.dropna()
    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), None) # Limitar Latitudes
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), None) # Limitar Longitudes

    return df

In [38]: def calcular_parametros_trayectoria(month_separated_trajectories):
    # Definir columnas del DataFrame de salida
    traj_parameters = ['split_id', 'traj_dist',
                       'traj_time', 'traj_mean_speed', 'traj_max_speed',
```

```

        'traj_mean_acc', 'traj_mean_dec']

# Agregar columnas de rangos de velocidad
speed_ranges_list = [
    [2.5, 3],
    [2, 4], [12, 14, 16, 18, 20], [0, 5], [10, 15, 20, 300]
]

for sr_index, speed_ranges in enumerate(speed_ranges_list, start=0):
    for i in range(len(speed_ranges) - 1):
        col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
        traj_parameters.append(col_name)

df_traj_parameters = cudf.DataFrame(columns=traj_parameters) # Crear un DataFrame vacío en cuDF
grouped = month_separated_trajectories.groupby('split_id') # Agrupar por trayectorias
df_traj_parameters['split_id'] = grouped['split_id'].first()

# Calcular distancia total y tiempo entre mensajes en segundos
df_traj_parameters['traj_dist'] = grouped['distance'].sum()
df_traj_parameters['traj_time'] = grouped['timestamp'].diff().fillna(0) / 1e9

# Para cada conjunto de rangos de velocidad, crear una columna con el porcentaje de datos que caen en ese tramo
for sr_index, speed_ranges in enumerate(speed_ranges_list, start=0):
    for i in range(len(speed_ranges) - 1):
        col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
        df_traj_parameters[col_name] = grouped.apply(
            lambda g: (
                ((g['speed'] >= speed_ranges[i]) & (g['speed'] < speed_ranges[i+1])).sum()
            ) / len(g)
        ) # La variable g, hace referencia al subconjunto del DataFrame que comparte el mismo valor de 'split_id'

# Calcular aceleraciones y deceleraciones en GPU
df_traj_parameters['traj_mean_dec'] = grouped['speed'].diff().min()
return df_traj_parameters

In [39]: def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories):
# Lista de parámetros a calcular
parameters = ['mmsi',
              'month_max_lat', 'month_min_lat', 'month_mean_lat',
              'month_max_lon', 'month_min_lon', 'month_mean_lon',
              'month_max_dif_lat', 'month_max_dif_lon',
              'traj_mean_dec_mean', 'traj_dist_max',
              # Rangos de velocidad (Range 1)
              'traj_percent_2dec5_3_mean',
              # Rangos de velocidad (Range 3)
              'traj_percent_2_4_mean', 'traj_percent_12_14_mean',
              'traj_percent_14_16_mean', 'traj_percent_16_18_mean',
              'traj_percent_18_20_mean',
              # Rango de velocidad (Range 4)
              'traj_percent_0_5_mean', 'traj_percent_10_15_mean',
              'traj_percent_15_20_mean', 'traj_percent_over20_mean'
              ]

# Crear un DataFrame vacío optimizado para GPU
df_parameters_temp = cudf.DataFrame(columns=parameters)

mmsi = month_trajectories['mmsi'].iloc[0]

# Parámetros geográficos
month_max_lat = month_trajectories['lat'].max()
month_min_lat = month_trajectories['lat'].min()
month_mean_lat = month_trajectories['lat'].mean()

month_max_lon = month_trajectories['lon'].max()
month_min_lon = month_trajectories['lon'].min()
month_mean_lon = month_trajectories['lon'].mean()

month_max_dif_lat = month_max_lat - month_min_lat
month_max_dif_lon = month_max_lon - month_min_lon

# Distancias
traj_dist_max = df_traj_parameters['traj_dist'].max()

# Rango de velocidades 1
traj_percent_2dec5_3_mean = df_traj_parameters['traj_percent_2.5-3'].mean()
# Rango de velocidades 3
traj_percent_2_4_mean = df_traj_parameters['traj_percent_2-4'].mean()
traj_percent_12_14_mean = df_traj_parameters['traj_percent_12-14'].mean()
traj_percent_14_16_mean = df_traj_parameters['traj_percent_14-16'].mean()
traj_percent_16_18_mean = df_traj_parameters['traj_percent_16-18'].mean()
traj_percent_18_20_mean = df_traj_parameters['traj_percent_18-20'].mean()
# Rango de velocidades 4
traj_percent_0_5_mean = df_traj_parameters['traj_percent_0-5'].mean()
traj_percent_10_15_mean = df_traj_parameters['traj_percent_10-15'].mean()
traj_percent_15_20_mean = df_traj_parameters['traj_percent_15-20'].mean()
traj_percent_over20_mean = df_traj_parameters['traj_percent_20-300'].mean()

# Aceleraciones y deceleraciones
traj_mean_dec_mean = df_traj_parameters['traj_mean_dec'].mean()

# Agregar los resultados al DataFrame
df_parameters_temp = cudf.DataFrame({
    'mmsi': [mmsi],
    'month_max_lat': [month_max_lat],
    'month_min_lat': [month_min_lat],
    'month_mean_lat': [month_mean_lat],
    'month_max_lon': [month_max_lon],
    'month_min_lon': [month_min_lon],
    'month_mean_lon': [month_mean_lon],
    'month_max_dif_lat': [month_max_dif_lat],
    'month_max_dif_lon': [month_max_dif_lon],
    'traj_dist_max': [traj_dist_max],

```

```
if 'shiptype' in df_merged_dynamic.columns:
    shiptype = df_merged_dynamic['shiptype'].iloc[0]
else:
    shiptype = 'unknown'
else:
    containing_nan += 1
else:
    too_short_csv += 1
else:
    non_associated_shiptypes += 1

total_time += time.time() - start_time
tiempos_procesamiento = cudf.concat([tiempos_procesamiento, cudf.DataFrame({'mmsi': [row['mmsi']], 'tiempo_procesamiento': [time.time() - sta

# Fusionar los resultados con el DataFrame original basado en 'mmsi'
df_cinematic_parameters = df_cinematic_parameters.merge(df_cinematic_results, on='mmsi', how='left')

print("Existen", containing_nan, "MMSI cuyo df contiene NaN.")
print("Existen", too_short_csv, "MMSI que no cumplen con la longitud requerida.")
print("Existen", non_associated_shiptypes, "MMSI que no se han asociado a ningún shiptype.")
print("Tiempo total en extraer parámetros:", total_time, "segundos.")

Processing MMSIs: 58%|██████████| 23764/40859 [3:11:10<1:48:47, 2.62it/s]
⚠ Se detectó un IndexError en la iteración 23764, omitiendo...
Processing MMSIs: 100%|██████████| 40859/40859 [5:18:05<00:00, 2.14it/s]
Existen 0 MMSI cuyo df contiene NaN.
Existen 3719 MMSI que no cumplen con la longitud requerida.
Existen 174 MMSI que no se han asociado a ningún shiptype.
Tiempo total en extraer parámetros: 18844.739042520523 segundos.

In [41]: df_cinematic_parameters.to_csv('cinematic_parameters_3_jan_XG.csv', index=False)
```


ANEXO IV: CARGA Y PROCESADO UNIFICADO DE LOS DATOS

```
In [1]: import csv
# Para Leer y escribir archivos CSV
import os
# Para manipular archivos y directorios
import pandas as pd
# Para manipulación y análisis de datos

import numpy as np
# Para computación científica y operaciones matemáticas
from tqdm import tqdm
# Para mostrar barras de progreso en bucles iterativos
import seaborn as sns
# Para visualización de datos estadísticos
import matplotlib.pyplot as plt
# Para visualización de datos en gráficos
from sklearn.model_selection import train_test_split, GridSearchCV, GroupShuffleSplit
# Para dividir conjuntos de datos y búsqueda de hiperparámetros
from sklearn.preprocessing import MinMaxScaler
# Para escalar características
from sklearn.metrics import classification_report
# Para evaluar el rendimiento del modelo
from sklearn.ensemble import RandomForestClassifier
# Para entrenar modelos de clasificación de bosques aleatorios
from sklearn.pipeline import Pipeline
# Para encadenar pasos de procesamiento y modelado
import hvplot.pandas
# Para visualización interactiva de pandas DataFrames
from shapely.geometry import Point
# Para trabajar con geometría en Python
from datetime import timedelta
# Para manipular fechas y tiempos
import geoviews as gv
# Para visualización de datos geoespaciales
from ydata_profiling import ProfileReport
# Para generar informes de perfil de datos
import holoviews as hv
# Para visualización interactiva de datos
from geopy.distance import great_circle, geodesic
# Para calcular distancias geoespaciales
from shapely.geometry import MultiPoint, LineString, Polygon
# Para trabajar con geometría en Python
from matplotlib.colors import LinearSegmentedColormap
# Para personalizar mapas de colores en Matplotlib
import xarray as xr
# Para trabajar con estructuras de datos multidimensionales
import cartopy.crs as ccrs
# Para visualización de mapas utilizando Cartopy
import plotly.express as px
# Para visualización interactiva de datos
import time
# Para medir el tiempo de ejecución de código
import cudf
import cupy as cp
import hvplot.cudf
from datetime import timedelta
import datetime

## Random Forest
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier # CPU
from sklearn.metrics import classification_report

In [2]: import torch
print(torch.cuda.current_device())
torch.cuda.set_device(1)
print(torch.cuda.current_device())

0
1

In [3]: def clean(df):
    df['mmsi'] = df['mmsi'].astype('int32') # ID del buque, ocupa menos espacio
    df['timestamp'] = df['timestamp'].astype('int64') # Mejor formato para cálculos de tiempo
    df['lat'] = df['lat'].astype('float32') # Precisión suficiente para coordenadas
    df['lon'] = df['lon'].astype('float32')
    df['speed'] = df['speed'].astype('float32') # Precisión suficiente para velocidad
    df['course'] = df['course'].astype('float32')
    df['heading'] = df['heading'].astype('int16') # Enteros pequeños ahorran memoria
    # Mapeo de shiptype a valores numéricos
    shiptype_mapping = {
        'ShipType.Cargo': 0,
        'ShipType.Fishing': 1,
        'ShipType.Passenger': 2,
        'ShipType.Tanker': 3,
        'ShipType.Tug': 4
    }
    # Reemplazar valores con el mapping y convertir a int8
    df['shiptype'] = df['shiptype'].map(shiptype_mapping).fillna(-1).astype('int8')

    df = df.sort_values(by=['mmsi', 'timestamp']) # Ordenar datos en GPU

    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), None) # Limitar Latitudes
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), None) # Limitar Longitudes
```

```

counts = df.groupby('mmsi').size().reset_index(name='count')
valid_mmsi = counts[counts['count'] >= 100]['mmsi']
df = df[df['mmsi'].isin(valid_mmsi)]

df.reset_index(inplace=True)
df = df.drop('Unnamed: 0', axis=1)
df = df.drop(columns=['index'])

return df

```

```

In [4]: def haversine_gpu(df):
        R = 6371000 # Radio de La Tierra en metros

        # Convertir Latitudes y Longitudes a radianes directamente en cuDF
        phi1 = cp.radians(df['lat'])
        phi2 = cp.radians(df['lat_shift'])
        delta_phi = cp.radians(df['lat_shift'] - df['lat'])
        delta_lambda = cp.radians(df['lon_shift'] - df['lon'])

        # Optimizar cálculos evitando repeticiones
        cos_phi1 = cp.cos(phi1)
        cos_phi2 = cp.cos(phi2)
        sin_delta_phi = cp.sin(delta_phi / 2.0) ** 2
        sin_delta_lambda = cp.sin(delta_lambda / 2.0) ** 2

        # Fórmula de Haversine
        a = sin_delta_phi + cos_phi1 * cos_phi2 * sin_delta_lambda
        c = 2 * cp.arctan2(cp.sqrt(a), cp.sqrt(1 - a))

        # Asignar resultado optimizado
        df = df.assign(distance=R * c)

        return df

In [5]: def calcular_trayectorias(df):
        df = cudf.DataFrame(df) # Asegurarse de que es un cuDF DataFrame
        DISTANCE_THRESHOLD = 10000 # Umbral de distancia en metros

        # Eliminar filas con distancias no reales
        df = df[df['distance'] <= 10e4]

        # Calcular velocidad en km/h
        df['velocidad'] = (df['distance'] * 3600) / (df['time_diff'] * 1000)

        # Identificar paradas (velocidad menor a 1 km/h)
        df['is_stop'] = (df['velocidad'] < 1).astype('int8')

        # Eliminar filas con velocidades ilógicas
        df = df[df['velocidad'] < 40]

        # Detectar transiciones de parada a movimiento
        df['transitions'] = ((df['is_stop'].shift(1) == 1) & (df['is_stop'] == 0)) | (df['mmsi'].shift(1) != df['mmsi'])

        # Contar transiciones acumuladas para segmentar trayectorias
        df['trajectory_id'] = df['transitions'].cumsum().astype('int32')

        # Calcular la distancia acumulada dentro de cada trayectoria
        df['dist_acum'] = df.groupby('trajectory_id')['distance'].cumsum().astype('int32')
        df['tiempo_acum'] = df.groupby('trajectory_id')['time_diff'].cumsum().astype('int32')

        # Crear un filtro para las trayectorias con dist_acum < DISTANCE_THRESHOLD
        low_distance_ids = df.groupby('trajectory_id').tail(1)
        low_distance_ids = df.groupby('trajectory_id')['dist_acum'] < DISTANCE_THRESHOLD['trajectory_id'].unique()

        # Asignar is_stop = 1 a las filas con esos trajectory_id
        df['is_stop'] = df['is_stop'].where(~df['trajectory_id'].isin(low_distance_ids), 1)

        # Eliminar valores aislados de is_stop (0,0,0,0,0,1,0,0,0 -> 0)
        df['prev'] = df['is_stop'].shift(1).fillna(0)
        df['next'] = df['is_stop'].shift(-1).fillna(0)
        df['is_stop'] = df['is_stop'].where((df['prev'] == 1) | (df['next'] == 1), 0)

        # Detectar transiciones de parada a movimiento
        df['transitions'] = ((df['is_stop'].shift(1) == 1) & (df['is_stop'] == 0)) | (df['is_stop'].shift(1) == 0) &
            (df['is_stop'] == 1) | (df['mmsi'].shift(1) != df['mmsi'])

        # Contar transiciones acumuladas para segmentar trayectorias
        df['trajectory_id'] = df['transitions'].cumsum().astype('int32')
        # Calcular la distancia acumulada dentro de cada trayectoria
        df['dist_acum'] = df.groupby('trajectory_id')['distance'].cumsum().astype('int32')
        df['tiempo_acum'] = df.groupby('trajectory_id')['time_diff'].cumsum().astype('int32')

        # Eliminar columnas auxiliares
        df = df.drop(columns=['prev', 'next', 'transitions'])

        return df

In [6]: def calcular_parametros_trayectoria(month_separated_trajectories):
        # Definir columnas del DataFrame de salida
        speed_ranges_list = [
            [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5],
            [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
            [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
            [0, 5, 10, 15, 20, 300]
        ]
        traj_parameters = ['traj_dist', 'traj_time',
                           'traj_mean_speed', 'traj_max_speed',
                           'traj_mean_acc', 'traj_mean_dec']

```

```
# Agregar nombres de columnas para los rangos de velocidad
for speed_ranges in speed_ranges_list:
    for i in range(len(speed_ranges) - 1):
        col_name = f"traj_percent_{speed_ranges[i]}-{speed_ranges[i+1]}"
        traj_parameters.append(col_name)
# Agrupar por 'trajectory_id' usando cuDF para mejorar rendimiento
grouped = month_separated_trajectories.groupby('trajectory_id')

# Crear DataFrame de resultados
df_traj_parameters = cudf.DataFrame()
last_row = month_separated_trajectories.groupby('trajectory_id').tail(1)
df_traj_parameters['trajectory_id'] = last_row['trajectory_id']
df_traj_parameters['mmsi'] = last_row['mmsi']
df_traj_parameters['traj_dist'] = last_row['dist_acum']
df_traj_parameters['traj_time'] = last_row['tiempo_acum']

# Velocidades
# Calcular métricas
mean_speed = grouped['velocidad'].mean().reset_index()
max_speed = grouped['velocidad'].max().reset_index()

# Unir los cálculos con el DataFrame final
df_traj_parameters = df_traj_parameters.merge(mean_speed, on='trajectory_id', how='left', suffixes=('', '_mean'))
df_traj_parameters = df_traj_parameters.merge(max_speed, on='trajectory_id', how='left', suffixes=('', '_max'))

# Distribución de velocidades en GPU sin apply()
# Contar el tiempo total de cada trayectoria
total_time = grouped['time_diff'].sum().reset_index()
total_time = total_time.rename(columns={'time_diff': 'total_time'})

# Fusionar con df_traj_parameters para garantizar alineación
df_traj_parameters = df_traj_parameters.merge(total_time, on='trajectory_id', how='left')

# Calcular el porcentaje de tiempo en cada rango de velocidad
for speed_ranges in speed_ranges_list:
    for i in range(len(speed_ranges) - 1):
        col_name = f"traj_percent_time_{speed_ranges[i]}-{speed_ranges[i+1]}"

        # Filtrar las filas en el rango de velocidad
        mask = (month_separated_trajectories['velocidad'] >= speed_ranges[i]) & \
            (month_separated_trajectories['velocidad'] < speed_ranges[i+1])

        # Sumar el tiempo ('time_diff') dentro del rango para cada trayectoria
        time_in_range = month_separated_trajectories[mask].groupby('trajectory_id')['time_diff'].sum().reset_index()
        time_in_range = time_in_range.rename(columns={'time_diff': col_name})

        # Unir con df_traj_parameters para asegurar alineación
        df_traj_parameters = df_traj_parameters.merge(time_in_range, on='trajectory_id', how='left')

        # Calcular el porcentaje sobre el tiempo total de la trayectoria
        df_traj_parameters[col_name] = (df_traj_parameters[col_name] / df_traj_parameters['total_time']).fillna(0)

# Eliminar columna auxiliar
df_traj_parameters = df_traj_parameters.drop(columns=['total_time'])

# Calcular la diferencia de velocidad entre puntos consecutivos dentro de cada trayectoria
month_separated_trajectories['speed_diff'] = month_separated_trajectories.groupby('trajectory_id')['velocidad'].diff()

# Aceleración media (solo valores positivos de speed_diff)
mean_acc = month_separated_trajectories[month_separated_trajectories['speed_diff'] > 0] \
    .groupby('trajectory_id')['speed_diff'].mean().reset_index()
mean_acc = mean_acc.rename(columns={'speed_diff': 'traj_mean_acc'})

# Deceleración media (solo valores negativos de speed_diff, convertidos a positivos)
mean_dec = month_separated_trajectories[month_separated_trajectories['speed_diff'] < 0] \
    .groupby('trajectory_id')['speed_diff'].mean().reset_index()
mean_dec['speed_diff'] = mean_dec['speed_diff'].abs() # Convertir a positivo
mean_dec = mean_dec.rename(columns={'speed_diff': 'traj_mean_dec'})

# Fusionar con df_traj_parameters
df_traj_parameters = df_traj_parameters.merge(mean_acc, on='trajectory_id', how='left')
df_traj_parameters = df_traj_parameters.merge(mean_dec, on='trajectory_id', how='left')

# Reemplazar NaN con 0 si una trayectoria no tiene aceleraciones/deceleraciones
df_traj_parameters = df_traj_parameters.fillna(0)

return df_traj_parameters
```

```
In [7]: def calcular_parametros_paradas(month_stops):
R = 6371000 # Radio de La Tierra en metros

# Convertir Lat/Lon a radianes directamente en cuDF
month_stops['lat_rad'] = month_stops['lat'] * (cp.pi / 180)
month_stops['lon_rad'] = month_stops['lon'] * (cp.pi / 180)

# Desplazar Lat/Lon por barco (MMSI)
month_stops['lat_shift'] = month_stops.groupby('mmsi')['lat_rad'].shift().fillna(0)
month_stops['lon_shift'] = month_stops.groupby('mmsi')['lon_rad'].shift().fillna(0)

# Calcular diferencias de coordenadas
delta_lat = month_stops['lat_shift'] - month_stops['lat_rad']
delta_lon = month_stops['lon_shift'] - month_stops['lon_rad']

# Fórmula de Haversine en cuDF (sin cupy para evitar conversiones lentas)
a = (cp.sin(delta_lat / 2) ** 2 +
     cp.cos(month_stops['lat_rad']) * cp.cos(month_stops['lat_shift']) * cp.sin(delta_lon / 2) ** 2)

c = 2 * cp.arctan2(cp.sqrt(a), cp.sqrt(1 - a))
month_stops['stop_dist'] = R * c
```

```

# Agrupar por 'trajectory_id' y calcular estadísticas
grouped = month_stops.groupby('trajectory_id').agg({
    'lat': ['max', 'min', 'mean'],
    'lon': ['max', 'min', 'mean'],
    'timestamp': 'count',
    'time_diff': ['max', 'min', 'mean'],
    'dist_acum': 'max',
    'mmsi': 'min'
}).reset_index()

# Renombrar columnas correctamente
grouped.columns = [
    'trajectory_id', 'stop_max_lat', 'stop_min_lat', 'stop_mean_lat',
    'stop_max_lon', 'stop_min_lon', 'stop_mean_lon',
    'stop_count', 'time_diff_max', 'time_diff_min', 'time_diff_mean', 'stop_dist', 'mmsi'
]

# Agregar número de parada (stop_N)
grouped.insert(0, "stop_N", grouped.index + 1)

return grouped

```

In [8]:

```

def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories, df_month_stops):
    # Parámetros de geolocalización
    month_trajectories_grouped = month_trajectories.groupby('mmsi')

    # Calcular las estadísticas de latitudes y longitudes por mmsi
    month_max_lat = month_trajectories_grouped['lat'].max()
    month_min_lat = month_trajectories_grouped['lat'].min()
    month_mean_lat = month_trajectories_grouped['lat'].mean()

    month_max_lon = month_trajectories_grouped['lon'].max()
    month_min_lon = month_trajectories_grouped['lon'].min()
    month_mean_lon = month_trajectories_grouped['lon'].mean()

    month_max_dif_lat = month_max_lat - month_min_lat
    month_max_dif_lon = month_max_lon - month_min_lon

    # Distancias y tiempos
    traj_dist_max = df_traj_parameters.groupby('mmsi')['traj_dist'].max()
    traj_dist_min = df_traj_parameters.groupby('mmsi')['traj_dist'].min()
    traj_dist_mean = df_traj_parameters.groupby('mmsi')['traj_dist'].mean()

    traj_time_mean = df_traj_parameters.groupby('mmsi')['traj_time'].mean()

    # Velocidades
    traj_mean_speed_mean = df_traj_parameters.groupby('mmsi')['velocidad'].mean()
    traj_max_speed_max = df_traj_parameters.groupby('mmsi')['velocidad_max'].max()

    month_mean_speed = month_trajectories_grouped['speed'].mean()
    month_max_speed = month_trajectories_grouped['speed'].max()

    # Aceleraciones y deceleraciones
    traj_mean_acc_mean = df_traj_parameters.groupby('mmsi')['traj_mean_acc'].mean()
    traj_mean_dec_mean = df_traj_parameters.groupby('mmsi')['traj_mean_dec'].mean()

    # Parámetros de paradas
    stop_dist_max = df_month_stops.groupby('mmsi')['stop_dist'].max()
    stop_dist_min = df_month_stops.groupby('mmsi')['stop_dist'].min()
    stop_dist_mean = df_month_stops.groupby('mmsi')['stop_dist'].mean()

    stop_time_max = df_month_stops.groupby('mmsi')['time_diff_max'].max()
    stop_time_min = df_month_stops.groupby('mmsi')['time_diff_min'].min()
    stop_time_mean = df_month_stops.groupby('mmsi')['time_diff_mean'].mean()

    # Calcular porcentaje de velocidad
    speed_ranges = [(0, 0.5), (0.5, 1), (1, 1.5), (1.5, 2), (2, 2.5), (2.5, 3), (3, 3.5), (3.5, 4),
                    (4, 4.5), (4.5, 5), (5, 10), (10, 15), (15, 20), (20, 300),
                    (0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10),
                    (0, 2), (2, 4), (4, 6), (6, 8), (8, 10), (10, 12), (12, 14), (14, 16), (16, 18), (18, 20),
                    (0, 5)]

    # Crear una máscara para cada rango de velocidad
    speed_percentages = {}
    for low, high in speed_ranges:
        mask = (month_trajectories['speed'] >= low) & (month_trajectories['speed'] < high)
        speed_count = month_trajectories[mask].groupby('mmsi')['speed'].count()
        total_count = month_trajectories.groupby('mmsi')['speed'].count()
        speed_percentages[f"traj_percent_{low}_{high}_mean"] = (speed_count / total_count).fillna(0)

    # Crear el DataFrame de salida
    df_parameters_temp = cudf.DataFrame({
        'mmsi': month_trajectories_grouped['mmsi'].last(),
        'shiptype': month_trajectories_grouped['shiptype'].last(),
        'N_Trajectories': df_traj_parameters.groupby('mmsi')['trajectory_id'].nunique(),
        'month_max_lat': month_max_lat,
        'month_min_lat': month_min_lat,
        'month_mean_lat': month_mean_lat,
        'month_max_lon': month_max_lon,
        'month_min_lon': month_min_lon,
        'month_mean_lon': month_mean_lon,
        'month_max_dif_lat': month_max_dif_lat,
        'month_max_dif_lon': month_max_dif_lon,
        'traj_dist_max': traj_dist_max,
        'traj_dist_min': traj_dist_min,
        'traj_dist_mean': traj_dist_mean,
        'month_dist': month_trajectories_grouped['mmsi']['lat'].count(),
        'traj_time_mean': traj_time_mean,
        'traj_mean_speed_mean': traj_mean_speed_mean,
        'traj_max_speed_max': traj_max_speed_max,
        'month_mean_speed': month_mean_speed,
    })

```

MEJORA DE LA EFICIENCIA COMPUTACIONAL DE UN SISTEMA DE PREDICCIÓN DE TIPO DE BUQUE UTILIZANDO PARÁMETROS DE TRAYECTORIAS Y TÉCNICAS DE IA

```
'month_max_speed': month_max_speed,
'traj_mean_acc_mean': traj_mean_acc_mean,
'traj_mean_dec_mean': traj_mean_dec_mean,
'N_Stops': df_month_stops.groupby('mmsi')['trajectory_id'].nunique(),
'stop_dist_max': stop_dist_max,
'stop_dist_min': stop_dist_min,
'stop_dist_mean': stop_dist_mean,
'stop_time_max': stop_time_max,
'stop_time_min': stop_time_min,
'stop_time_mean': stop_time_mean,
**speed_percentages
})

return df_parameters_temp
```

```
In [9]: %%time
df_dynamic = cudf.read_csv('dynamic.csv')
df_dynamic = clean(df_dynamic)
df_dynamic['lat_shift'] = df_dynamic.groupby('mmsi')['lat'].shift().astype('float32')
df_dynamic['lon_shift'] = df_dynamic.groupby('mmsi')['lon'].shift().astype('float32')
df_dynamic['time_diff'] = df_dynamic.groupby('mmsi')['timestamp'].diff().astype('int64')
df_dynamic=df_dynamic.dropna()
df_dynamic=haversine_gpu(df_dynamic)

#Eliminar valores nulos
df_dynamic = df_dynamic.dropna()
df_dynamic = calcular_trayectorias(df_dynamic)
df_paradas = df_dynamic[df_dynamic['is_stop']==1]
df_trayectorias = df_dynamic[df_dynamic['is_stop']==0]
parametros_trayectorias = calcular_parametros_trayectoria(df_trayectorias)
parametros_paradas = calcular_parametros_paradas(df_paradas)
parametros_cinematicos = calcular_parametros_cinematicos(parametros_trayectorias, df_dynamic, parametros_paradas)
parametros_cinematicos = parametros_cinematicos.fillna(0)

CPU times: user 11.4 s, sys: 4.68 s, total: 16 s
Wall time: 10.6 s
```

In [11]: parametros_cinematicos

```
Out[11]:
```

	mmsi	shiptype	N_Trajectories	month_max_lat	month_min_lat	month_mean_lat	month_max_lon	month_min_lon	month_mean_lon	month_max_d
0	5	4	7	9.617068	-21.800053	8.195235	124.759834	-173.696609	120.697767	31.41
1	6	4	5	9.016103	-21.934406	8.016463	124.759880	-173.472229	119.047298	30.95
2	2829	0	8	-25.544207	-29.230938	-27.035780	-57.565269	-59.438766	-58.462974	3.66
3	3247	2	25	12.593527	11.925053	12.568184	121.948708	121.531662	121.545455	0.66
4	8086	2	27	9.887363	9.810703	9.819873	125.483871	125.299957	125.450131	0.07
...	...	...	...	...	...	...	...	...	...	...
38431	827002226	4	1	13.075975	13.047183	13.056420	100.905853	100.847847	100.888443	0.02
38432	848290500	0	1	14.625908	14.623703	14.624212	120.875465	120.873306	120.874880	0.00
38433	900004001	0	18	11.009773	10.313645	10.355668	124.597679	123.939156	123.981620	0.69
38434	924467387	1	0	51.516239	51.484604	51.499124	4.094505	4.049852	4.055687	0.03
38435	985524411	4	0	-8.772572	-8.772663	-8.772597	13.253562	13.253447	13.253446	0.00

38436 rows x 64 columns

```
In [10]: parametros_cinematicos=parametros_cinematicos.drop('mmsi',axis=1)
parametros_cinematicos=parametros_cinematicos.reset_index(drop=True)
parametros_cinematicos.to_csv("parametros_cinematicos.csv")
```

```
In [16]: print(parametros_cinematicos.isnull().any().any())

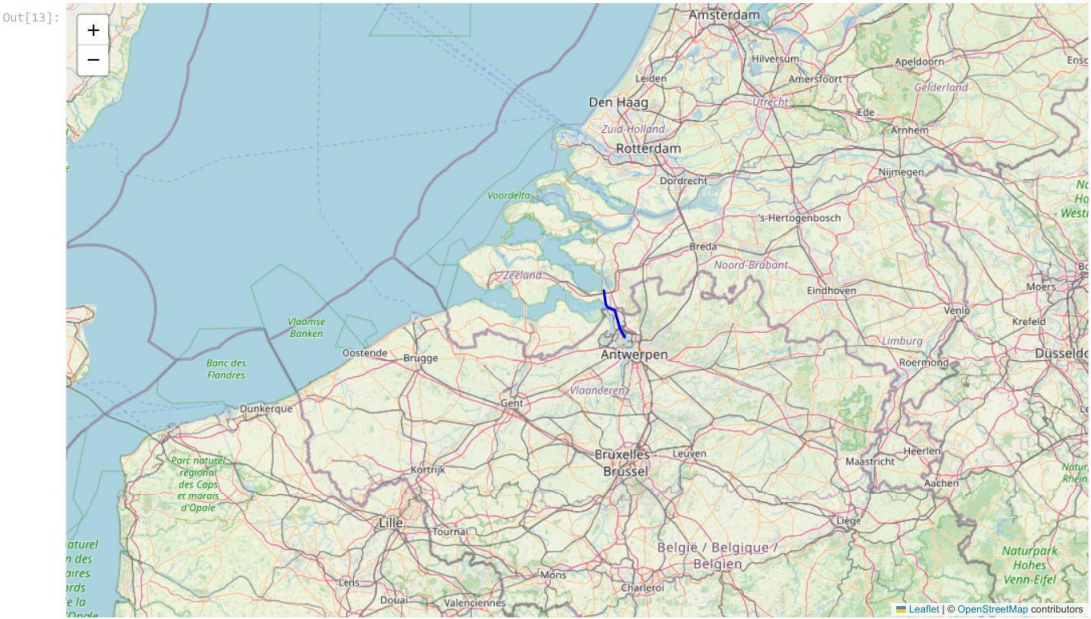
False
```

```
In [13]: import folium
import cudf
import pandas as pd
#d3=d2[d2['mmsi']==205220390]
d3=df_dynamic[df_dynamic['trajectory_id']==177603]
d3=d3.reset_index()
#d2=d2[d2['trajectory_id']==0]
# Crear un mapa centrado en la primera posición del trayecto
mapa = folium.Map(location=[d3['lat'][0], d3['lon'][0]], zoom_start=8)

# Crear una Lista de las coordenadas (Latitud, Longitud) para la trayectoria
coordinates = [(d3['lat'][i], d3['lon'][i]) for i in range(len(d3))]

# Añadir la trayectoria como una línea (Polilínea) en el mapa
folium.Polyline(coordinates, color='blue', weight=2.5, opacity=1).add_to(mapa)

# Para visualizar el mapa en un entorno de Jupyter, simplemente ejecuta la variable 'mapa'
```



ANEXO V: RESULTADOS EN LA EVALUACIÓN DE LOS MODELOS

```
In [8]: from cuml.model_selection import train_test_split # o usa otra alternativa para dividir en GPU
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
import cudf
import cupy as cp
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

In [54]: # Cargar con Pandas
df_final_static_pd = pd.read_csv("final_static_data_jan.csv")
df_cinematic_parameters_pd = pd.read_csv("cinematic_parameters_1_jan.csv")
df_cinematic_parameters_pd = df_cinematic_parameters_pd.drop(columns=['mmsi'])

In [55]: # Experimento 1: RANDOM FOREST - DINAMICOS
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_cinematic_parameters_pd, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline (CPU y scikit-Learn)
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47, n_jobs=-1)) # n_jobs=-1 usa todos los núcleos de la CPU
])

# Entrenar el modelo en CPU
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo Random Forest: {accuracy:.4f}")

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=0))

# Obtener el modelo RandomForest dentro del Pipeline
forest = model.named_steps['classifier']

# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# Crear un DataFrame con la importancia de cada variable
fi = pd.DataFrame({
    'att': x_train.columns,
    'importance': importances
}).sort_values(by='importance', ascending=False).reset_index(drop=True)

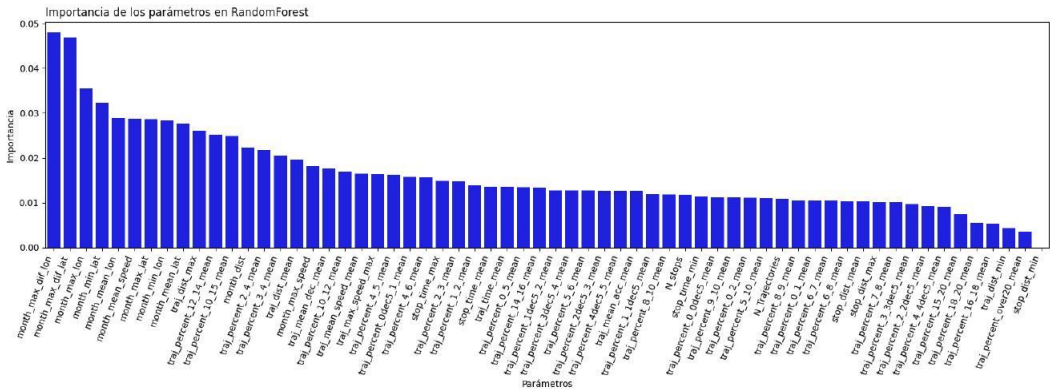
# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=fi, x='att', y='importance', color='b')

plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Parámetros")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest", loc="left")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()

Precisión del modelo Random Forest: 0.7540
      precision    recall  f1-score   support

0         0.74      0.91      0.82       4566
1         0.84      0.80      0.82       1104
2         0.81      0.55      0.66        596
3         0.70      0.35      0.46       1726
4         0.77      0.80      0.78       1214

accuracy          0.75       9206
macro avg         0.77      0.68      0.71       9206
weighted avg      0.75      0.75      0.74       9206
```



```
In [4]: from sklearn.ensemble import RandomForestClassifier

# Verificar qué criterio usa el modelo para evaluar nodos
print(f"Criterio usado en Random Forest: {forest.criterion}")
```

Criterio usado en Random Forest: gini

```
In [56]: df_final_static = pd.read_csv("final_static_data_jan.csv")
df_cinematic_parameters_1 = pd.read_csv('cinematic_parameters_1_jan.csv')
df_global_parameters_1 = pd.merge(df_cinematic_parameters_1, df_final_static, left_on='mmsi', right_on='MMSI', how='inner')
df_global_parameters_1 = df_global_parameters_1.drop(columns=['shiptype_y'])
df_global_parameters_1 = df_global_parameters_1.rename(columns={'shiptype_x': 'shiptype'})
df_global_parameters_1 = df_global_parameters_1.drop(columns=['mmsi', 'MMSI'])
```

```

In [57]: # Experimento 2 - RANDOM FOREST - df_global_parameters_1
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_1, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47))
])

# Entrenar el modelo
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo Random Forest: {accuracy:.4f}")

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=1))

# Obtener el modelo RandomForest dentro del Pipeline
forest = model.named_steps['classifier']

# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# Crear un DataFrame con la importancia de cada variable
fi = pd.DataFrame({
    'att': x_train.columns,
    'importance': importances
}).sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=fi, x='att', y='importance', color='b')

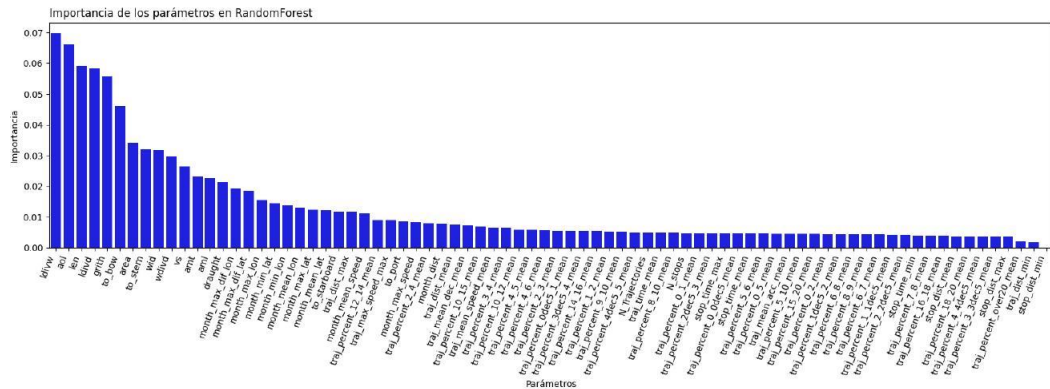
plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Parámetros")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest", loc="left")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()

```

MEJORA DE LA EFICIENCIA COMPUTACIONAL DE UN SISTEMA DE PREDICCIÓN DE TIPO DE BUQUE UTILIZANDO PARÁMETROS DE TRAYECTORIAS Y TÉCNICAS DE IA

Precisión del modelo Random Forest: 0.8958

	precision	recall	f1-score	support
0	0.89	0.95	0.92	4566
1	0.89	0.93	0.91	1184
2	0.86	0.80	0.83	596
3	0.92	0.75	0.83	1726
4	0.90	0.90	0.90	1214
accuracy			0.90	9206
macro avg	0.89	0.87	0.88	9206
weighted avg	0.90	0.90	0.89	9206



```
In [7]: # PRIMERA PRUEBA PRELIMINAR SEGUNDA ITERACION
columnas_a_eliminar = ['stop_dist_min', 'traj_dist_min', 'stop_dist_max',
                        'stop_dist_mean', 'stop_time_min', 'stop_time_max',
                        'stop_time_mean', 'N_trajectories', 'N_stops',
                        'traj_percent_over20_mean', 'traj_percent_4_4dec5_mean',
                        'traj_percent_2_2dec5_mean', 'traj_percent_6_8_mean',
                        'traj_percent_3_3dec5_mean', 'traj_percent_6_7_mean',
                        'traj_percent_7_8_mean', 'traj_percent_15_20_mean',
                        'traj_percent_8_9_mean', 'traj_percent_0_2_mean',
                        'traj_percent_1_1dec5_mean', 'traj_percent_16_18_mean',
                        'traj_percent_3dec5_4_mean', 'traj_percent_1dec5_2_mean',
                        'traj_percent_18_20_mean', 'traj_percent_0_0dec5_mean',
                        'traj_mean_acc_mean',
                        'traj_percent_5_10_mean', 'traj_percent_0_1_mean',
                        'traj_percent_0_5_mean']

# Eliminar las columnas especificadas, ignorando errores si alguna no existe
df_cinematic_parameters_pd = df_cinematic_parameters_pd.drop(columns=columnas_a_eliminar, errors='ignore')

In [8]: # Eliminar las columnas especificadas, ignorando errores si alguna no existe
df_cinematic_parameters_pd = df_cinematic_parameters_pd.drop(columns=columnas_a_eliminar, errors='ignore')

In [9]: # PRIMERA PRUEBA PRELIMINAR (Se reducen Los parámetros extraídos): RANDOM FOREST - df_cinematic_parameters_2

#df_cinematic_parameters = df_cinematic_parameters.to_pandas()
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_cinematic_parameters_pd, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline (CPU y scikit-Learn)
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47, n_jobs=-1)) # n_jobs=-1 usa todos Los núcleos de La CPU
])

# Entrenar el modelo en CPU
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo Random Forest: {accuracy:.4f}")

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=0))

# Obtener el modelo RandomForest dentro del Pipeline
forest = model.named_steps['classifier']

# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# Crear un DataFrame con La importancia de cada variable
fi = pd.DataFrame({
```

```

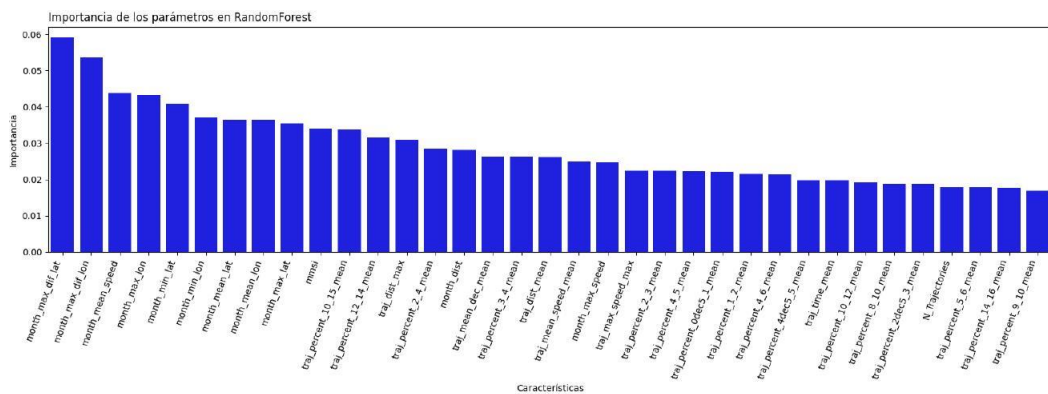
'att': x_train.columns,
'importance': importances
)).sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=fi, x='att', y='importance', color='b')

plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Características")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest", loc="left")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()

```

	precision	recall	f1-score	support
0	0.75	0.91	0.82	4566
1	0.84	0.82	0.83	1104
2	0.81	0.57	0.67	596
3	0.70	0.38	0.49	1726
4	0.77	0.78	0.78	1214
accuracy			0.76	9206
macro avg	0.77	0.69	0.72	9206
weighted avg	0.76	0.76	0.74	9206



```

In [10]: # EXPERIMENTO 3 (Segunda iteracion) Se extraen Los parámetros seleccionados.
df_cinematic_parameters_2_RF = pd.read_csv('cinematic_parameters_2_jan_RF.csv')
df_cinematic_parameters_2_RF = df_cinematic_parameters_2_RF.drop(columns=['mmsi'])

```

```

In [11]: print(df_cinematic_parameters_2_RF.columns)

```

```

Index(['shiptype', 'month_max_lat', 'month_min_lat', 'month_mean_lat',
      'month_max_lon', 'month_min_lon', 'month_mean_lon', 'month_max_dif_lat',
      'month_max_dif_lon', 'traj_dist_max', 'traj_dist_mean', 'month_dist',
      'traj_time_mean', 'traj_mean_speed_mean', 'traj_max_speed_max',
      'month_mean_speed', 'month_max_speed', 'traj_mean_dec_mean',
      'traj_percent_0dec5_1_mean', 'traj_percent_2dec5_3_mean',
      'traj_percent_4dec5_5_mean', 'traj_percent_1_2_mean',
      'traj_percent_2_3_mean', 'traj_percent_3_4_mean',
      'traj_percent_4_5_mean', 'traj_percent_5_6_mean',
      'traj_percent_9_10_mean', 'traj_percent_2_4_mean',
      'traj_percent_4_6_mean', 'traj_percent_8_10_mean',
      'traj_percent_10_12_mean', 'traj_percent_12_14_mean',
      'traj_percent_14_16_mean', 'traj_percent_10_15_mean'],
      dtype='object')

```

```

In [12]: print(df_cinematic_parameters_2_RF.shape)

```

```

(36825, 34)

```

```

In [13]: # PRIMERA PRUEBA PRELIMINAR
# Se reducen los parámetros de la primera prueba preliminar: RANDOM FOREST - df_cinematic_parameters_2

#df_cinematic_parameters = df_cinematic_parameters.to_pandas()
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_cinematic_parameters_2_RF, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline (CPU y scikit-Learn)
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47, n_jobs=-1)) # n_jobs=-1 usa todos los núcleos de la CPU
])

# Entrenar el modelo en CPU
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Mostrar resultados

```

```
print(classification_report(y_test, y_pred, zero_division=0))

# Obtener el modelo RandomForest dentro del Pipeline
forest = model.named_steps['classifier']

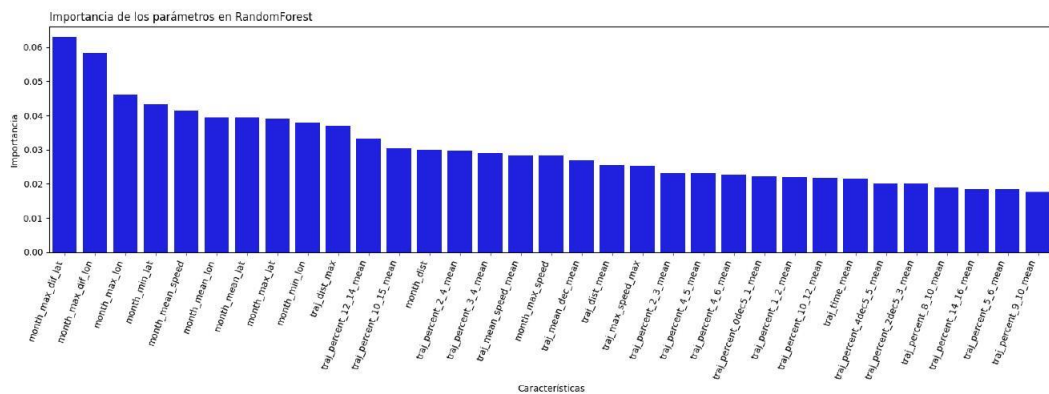
# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# Crear un DataFrame con la importancia de cada variable
fi = pd.DataFrame({
    'att': x_train.columns,
    'importance': importances
}).sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=fi, x='att', y='importance', color='b')

plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Características")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest", loc="left")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()
```

	precision	recall	f1-score	support
0	0.75	0.90	0.82	4571
1	0.80	0.79	0.80	1052
2	0.81	0.58	0.68	570
3	0.72	0.39	0.51	1791
4	0.78	0.79	0.79	1223
accuracy			0.76	9207
macro avg	0.77	0.69	0.72	9207
weighted avg	0.76	0.76	0.74	9207



```
In [14]: df_cinematic_parameters_2_RF = pd.read_csv('cinematic_parameters_2_jan_RF.csv')
df_final_static = pd.read_csv('final_static_data_jan.csv')
df_global_parameters_2_RF = pd.merge(df_cinematic_parameters_2_RF, df_final_static, left_on='mmsi', right_on='MMSI', how='inner')
df_global_parameters_2_RF = df_global_parameters_2_RF.drop(columns=['shiptype_y', 'mmsi', 'MMSI'])
df_global_parameters_2_RF = df_global_parameters_2_RF.rename(columns={'shiptype_x': 'shiptype'})
```

```
In [15]: # PRIMERA PRUEBA PRELIMINAR
# Parámetros de la primera prueba preliminar: df_global_parameters_2_RF
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_2_RF, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47))
])

# Entrenar el modelo
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=1))

# Obtener el modelo RandomForest dentro del Pipeline
forest = model.named_steps['classifier']

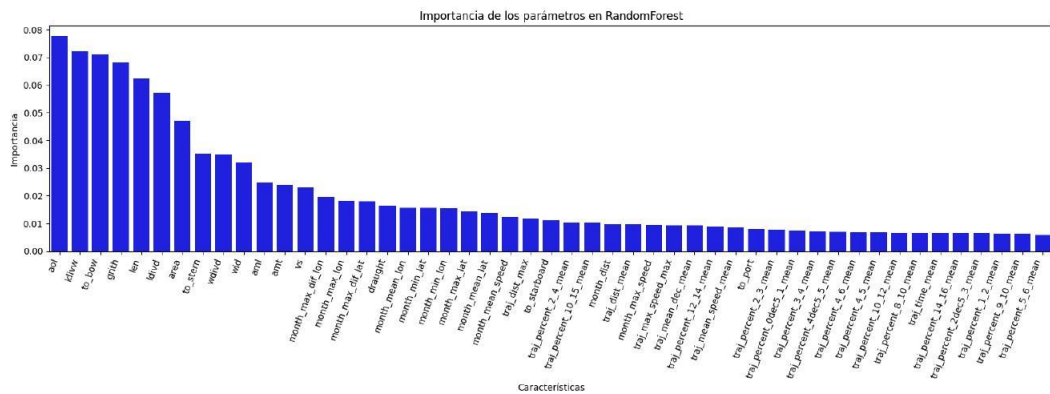
# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
```

```
# Crear un DataFrame con la importancia de cada variable
fi = pd.DataFrame({
    'att': x_train.columns,
    'importance': importances
}).sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=fi, x='att', y='importance', color='b')

plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Características")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()
```

	precision	recall	f1-score	support
0	0.90	0.95	0.92	4571
1	0.89	0.91	0.90	1052
2	0.85	0.85	0.85	570
3	0.91	0.77	0.83	1791
4	0.92	0.91	0.92	1223
accuracy			0.90	9207
macro avg	0.89	0.88	0.88	9207
weighted avg	0.90	0.90	0.90	9207



```
In [17]: df_cinematic_parameters_2_RF = pd.read_csv('cinematic_parameters_2_jan_RF.csv')
df_global_parameters_2 = pd.merge(df_cinematic_parameters_2_RF, df_final_static, left_on='mmsi', right_on='MMSI', how='inner')
df_global_parameters_2 = df_global_parameters_2.drop(columns=['shiptype_y', 'mmsi', 'MMSI'])
df_global_parameters_2 = df_global_parameters_2.rename(columns={'shiptype_x': 'shiptype'})
```

```
In [18]: # SEGUNDA PRUEBA PRELIMINAR RANDOM FOREST
# eliminamos mas columnas
columnas_a_eliminar = ['traj_percent_5_6_mean', 'traj_percent_9_10_mean', 'traj_percent_1_2_mean',
                       'traj_percent_2dec5_3_mean', 'traj_percent_14_16_mean', 'traj_time_mean',
                       'traj_percent_8_10_mean', 'traj_percent_10_12_mean',
                       'traj_percent_1_2_mean', 'traj_percent_2dec5_3_mean', 'traj_percent_4dec5_5_mean',
                       'traj_percent_4_5_mean', 'traj_percent_3_4_mean',
                       'traj_percent_2_3_mean', 'traj_percent_0dec5_1_mean',
                       'traj_percent_4_6_mean']

# Eliminar las columnas especificadas, ignorando errores si alguna no existe
df_global_parameters_3_RF = df_global_parameters_2_RF.drop(columns=columnas_a_eliminar, errors='ignore')
```

```
In [19]: print(df_global_parameters_3_RF.shape)

(36825, 36)
```

```
In [20]: # SEGUNDA PRUEBA PRELIMINAR
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_3_RF, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47))
])

# Entrenar el modelo
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=1))

# Obtener el modelo RandomForest dentro del Pipeline
```

```
forest = model.named_steps['classifier']

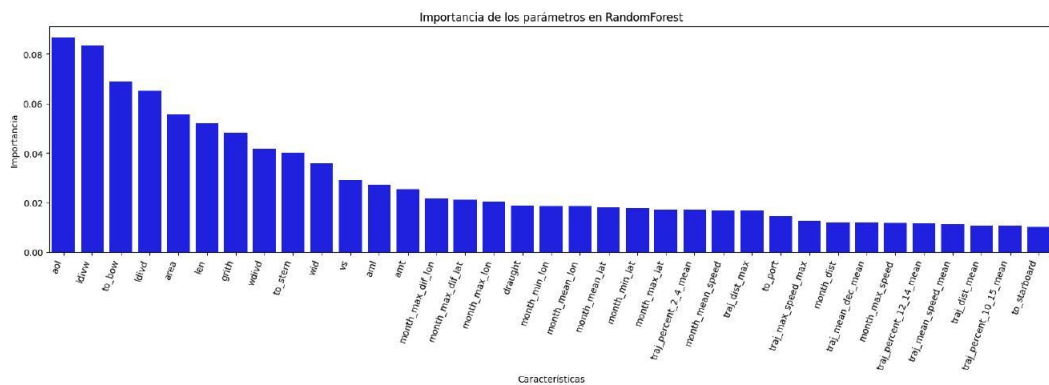
# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# Crear un DataFrame con la importancia de cada variable
fi = pd.DataFrame({
    'att': x_train.columns,
    'importance': importances
}).sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=fi, x='att', y='importance', color='b')

plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Características")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()
```

	precision	recall	f1-score	support
0	0.90	0.95	0.92	4571
1	0.89	0.91	0.90	1052
2	0.85	0.84	0.85	570
3	0.91	0.78	0.84	1791
4	0.91	0.90	0.91	1223
accuracy			0.90	9207
macro avg	0.89	0.88	0.88	9207
weighted avg	0.90	0.90	0.90	9207



```
In [50]: df_cinematic_parameters_3_RF = pd.read_csv('cinematic_parameters_3_jan_RF.csv')
df_cinematic_parameters_3_RF = df_cinematic_parameters_3_RF.drop(columns=['mmsi'])
```

```
In [51]: # Experimento 3: RANDOM FOREST - DINAMICOS
# Parámetros de la segunda prueba preliminar
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_cinematic_parameters_3_RF, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47))
])

# Entrenar el modelo
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo Random Forest: {accuracy:.4f}")

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=1))

# Obtener el modelo RandomForest dentro del Pipeline
forest = model.named_steps['classifier']

# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# Crear un DataFrame con la importancia de cada variable
fi = pd.DataFrame({
```

```

'att': x_train.columns,
'importance': importances
)).sort_values(by='importance', ascending=False).reset_index(drop=True)

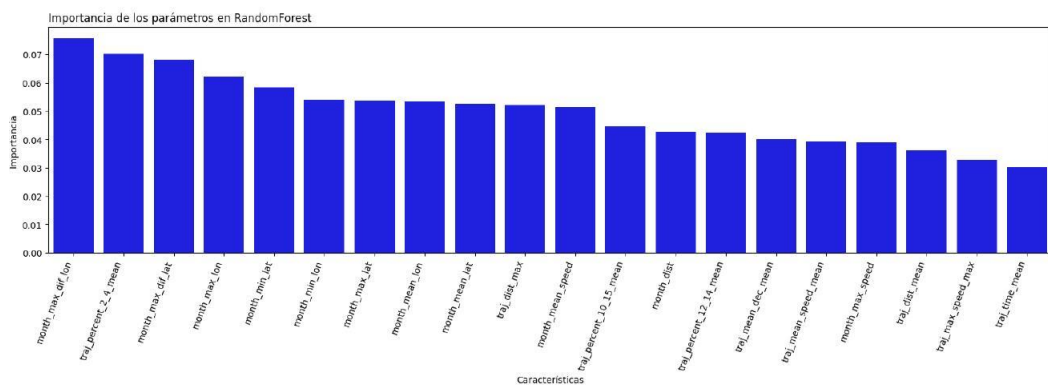
# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=fi, x='att', y='importance', color='b')

plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Características")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest", loc="left")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()

```

Precisión del modelo Random Forest: 0.7345

	precision	recall	f1-score	support
0	0.71	0.92	0.80	4972
1	0.82	0.74	0.77	1271
2	0.79	0.53	0.64	688
3	0.74	0.37	0.50	1934
4	0.76	0.69	0.72	1350
accuracy			0.73	10215
macro avg	0.76	0.65	0.69	10215
weighted avg	0.74	0.73	0.72	10215



```

In [52]: df_cinematic_parameters_3_RF = pd.read_csv('cinematic_parameters_3_jan_RF.csv')
df_final_static = pd.read_csv("final_static_data_jan.csv")
df_global_parameters_3_RF = pd.merge(df_cinematic_parameters_3_RF, df_final_static, left_on='mmsi', right_on='MMSI', how='inner')
df_global_parameters_3_RF = df_global_parameters_3_RF.drop(columns=['shiptype_y', 'mmsi', 'MMSI'])
df_global_parameters_3_RF = df_global_parameters_3_RF.rename(columns={'shiptype_x': 'shiptype'})

```

```

In [24]: df_global_parameters_3_RF.columns

```

```

Out[24]: Index(['shiptype', 'month_max_lat', 'month_min_lat', 'month_mean_lat',
'month_max_lon', 'month_min_lon', 'month_mean_lon', 'month_max_dif_lat',
'month_max_dif_lon', 'traj_dist_max', 'traj_dist_mean', 'month_dist',
'traj_time_mean', 'traj_mean_speed_mean', 'traj_max_speed_max',
'month_mean_speed', 'month_max_speed', 'traj_mean_dec_mean',
'traj_percent_2_4_mean', 'traj_percent_12_14_mean',
'traj_percent_10_15_mean', 'draught', 'to_bow', 'to_stern', 'to_port',
'to_starboard', 'len', 'wid', 'ldivw', 'ldivd', 'wdivd', 'area',
'grith', 'aml', 'amt', 'vs', 'aol'],
dtype='object')

```

```

In [53]: # Experimento 4: RANDOM FOREST - df_global_parameters_3_RF
# Parámetros de la segunda prueba preliminar
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_3_RF, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline
pipe = Pipeline([
('scaler', MinMaxScaler()), # Normalización de datos
('classifier', RandomForestClassifier(n_estimators=100, random_state=47))
])

# Entrenar el modelo
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo Random Forest: {accuracy:.4f}")

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=1))

# Obtener el modelo RandomForest dentro del Pipeline

```

precisión	recall	f1-score	support
0	0.90	0.95	4972
1	0.87	0.92	1271
2	0.86	0.83	688
3	0.91	0.78	1934
4	0.91	0.89	1350
accuracy		0.90	10215
macro avg	0.89	0.87	10215
weighted avg	0.90	0.90	10215



```
Out[27]: Index(['shiptype', 'N_Trajectories', 'month_max_lat', 'month_min_lat',
'month_mean_lat', 'month_max_lon', 'month_min_lon', 'month_mean_lon',
'month_max_dif_lat', 'month_max_dif_lon', 'traj_dist_max',
'traj_dist_min', 'traj_dist_mean', 'month_dist', 'traj_time_mean',
'traj_mean_speed_mean', 'traj_max_speed_max', 'month_mean_speed',
'month_max_speed', 'traj_mean_acc_mean', 'traj_mean_dec_mean',
'N_stops', 'stop_dist_max', 'stop_dist_min', 'stop_dist_mean',
'stop_time_max', 'stop_time_min', 'stop_time_mean',
'traj_percent_0_0dec5_mean', 'traj_percent_0dec5_1_mean',
'traj_percent_1_1dec5_mean', 'traj_percent_1dec5_2_mean',
'traj_percent_2_2dec5_mean', 'traj_percent_2dec5_3_mean',
'traj_percent_3_3dec5_mean', 'traj_percent_3dec5_4_mean',
'traj_percent_4_4dec5_mean', 'traj_percent_4dec5_5_mean',
'traj_percent_0_1_mean', 'traj_percent_1_2_mean',
'traj_percent_2_3_mean', 'traj_percent_3_4_mean',
'traj_percent_4_5_mean', 'traj_percent_5_6_mean',
'traj_percent_6_7_mean', 'traj_percent_7_8_mean',
'traj_percent_8_9_mean', 'traj_percent_9_10_mean',
'traj_percent_0_2_mean', 'traj_percent_2_4_mean',
'traj_percent_4_6_mean', 'traj_percent_6_8_mean',
'traj_percent_8_10_mean', 'traj_percent_10_12_mean',
'traj_percent_12_14_mean', 'traj_percent_14_16_mean',
'traj_percent_16_18_mean', 'traj_percent_18_20_mean',
'traj_percent_0_5_mean', 'traj_percent_5_10_mean',
'traj_percent_10_15_mean', 'traj_percent_15_20_mean',
'traj_percent_over20_mean'],
dtype='object')
```

135

```

df_train, df_test = train_test_split(df_cinematic_parameters_pd, test_size=0.2, random_state=43,
                                    stratify=df_cinematic_parameters_pd["shiptype"])

# Separar características y variable objetivo
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train["shiptype"]
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test["shiptype"]

# Creación del modelo con los datos correctos
x_train_dmatrix = xgb.DMatrix(x_train, label=y_train)
x_test_dmatrix = xgb.DMatrix(x_test, label=y_test)

# Definir parámetros
parametros = {
    "booster": "gbtree",
    "num_class": len(y_train.unique()),
    "max_depth": 10,
    "eta": 0.2,
    "objective": "multi:softmax",
    "nthread": 2
}

rondas = 650
modelo = xgb.train(parametros, x_train_dmatrix, rondas) # Entrenar el modelo
prediccion = modelo.predict(x_test_dmatrix) # Hacer predicciones

# Evaluar el modelo
accuracy = accuracy_score(y_test, prediccion)
print(f"Precisión del modelo: {accuracy:.4f}")

# Reporte de métricas
reporte = classification_report(y_test, prediccion, digits=2)
print("\nReporte de Clasificación:\n", reporte)

# Obtén la importancia de las características usando 'gain'
importance_dict = modelo.get_score(importance_type='gain')

# Convierte el diccionario a DataFrame
fi = pd.DataFrame(list(importance_dict.items()), columns=['att', 'importance'])
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)

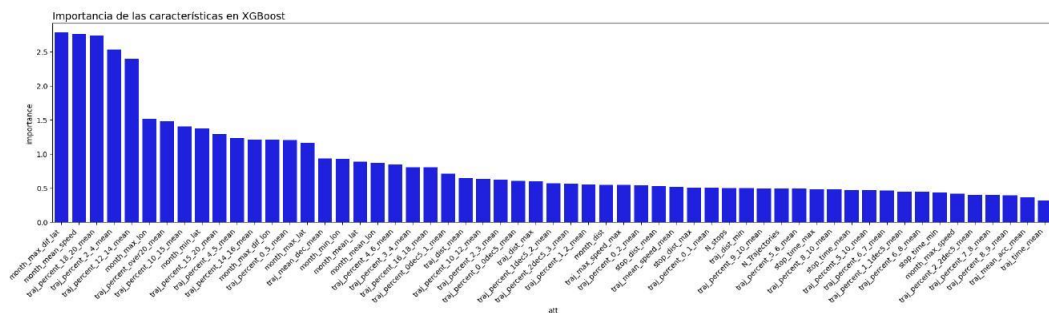
# Graficar la importancia usando seaborn
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(25, 5) # Ajustar el tamaño del gráfico
plt.xticks(rotation=45, ha='right') # Rotar etiquetas y alinearlas correctamente
plt.title("Importancia de las características en XGBoost", fontsize=14, loc='left')
plt.show() # Mostrar el gráfico

```

Precisión del modelo: 0.7973

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.80	0.90	0.85	3650
1	0.86	0.84	0.85	886
2	0.78	0.63	0.69	466
3	0.75	0.56	0.64	1424
4	0.79	0.82	0.81	939
accuracy			0.80	7365
macro avg	0.80	0.75	0.77	7365
weighted avg	0.79	0.80	0.79	7365



```

In [61]: df_final_static = pd.read_csv("final_static_data_jan.csv")
df_cinematic_parameters_1 = pd.read_csv('cinematic_parameters_1_jan.csv')
df_global_parameters_1 = pd.merge(df_cinematic_parameters_1, df_final_static, left_on='mmsi', right_on='MMSI', how='inner')
df_global_parameters_1 = df_global_parameters_1.drop(columns=['shiptype_y'])
df_global_parameters_1 = df_global_parameters_1.rename(columns={'shiptype_x': 'shiptype'})
df_global_parameters_1 = df_global_parameters_1.drop(columns=['mmsi', 'MMSI'])

```

In [32]: # EXPERIMENTO 2: XGBOOST - df_global_parameters_1

```

import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder

# Asegurar que shiptype sea numérico
encoder = LabelEncoder()
df_global_parameters_1["shiptype"] = encoder.fit_transform(df_global_parameters_1["shiptype"])

```

MEJORA DE LA EFICIENCIA COMPUTACIONAL DE UN SISTEMA DE PREDICCIÓN DE TIPO DE BUQUE UTILIZANDO PARÁMETROS DE TRAYECTORIAS Y TÉCNICAS DE IA

```
# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_1, test_size=0.2, random_state=43,
                                     stratify=df_global_parameters_1["shiptype"])

# Separar características y variable objetivo
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train["shiptype"]
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test["shiptype"]

# Creación del modelo con los datos correctos
x_train_dmatrix = xgb.DMatrix(x_train, label=y_train)
x_test_dmatrix = xgb.DMatrix(x_test, label=y_test)

# Definir parámetros
parametros = {
    "booster": "gbtree",
    "num_class": len(y_train.unique()),
    "max_depth": 10,
    "eta": 0.2,
    "objective": "multi:softmax",
    "nthread": 2
}

rondas = 650
modelo = xgb.train(parametros, x_train_dmatrix, rondas) # Entrenar el modelo
prediccion = modelo.predict(x_test_dmatrix) # Hacer predicciones

# Evaluar el modelo
accuracy = accuracy_score(y_test, prediccion)
print(f"Precisión del modelo: {accuracy:.4f}")

# Reporte de métricas
reporte = classification_report(y_test, prediccion, digits=2)
print("\nReporte de Clasificación:\n", reporte)

# Obtén la importancia de las características usando 'gain'
importance_dict = modelo.get_score(importance_type='gain')

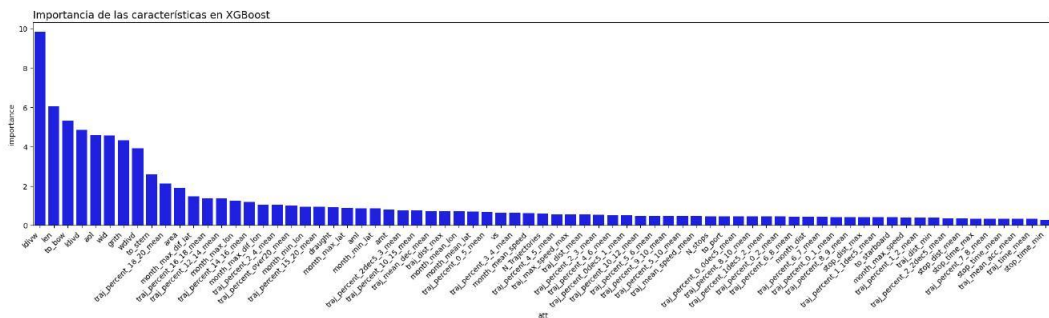
# Convierte el diccionario a DataFrame
fi = pd.DataFrame(list(importance_dict.items()), columns=['att', 'importance'])
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar la importancia usando seaborn
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(25, 5) # Ajustar el tamaño del gráfico
plt.xticks(rotation=45, ha='right') # Rotar etiquetas y alinearlas correctamente
plt.title("Importancia de las características en XGBoost", fontsize=14, loc="left") # Agregar título con "XGBoost" en cursiva
plt.show() # Mostrar el gráfico
```

Precisión del modelo: 0.9139

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.92	0.95	0.93	3650
1	0.92	0.92	0.92	886
2	0.86	0.86	0.86	466
3	0.91	0.83	0.87	1424
4	0.92	0.92	0.92	939
accuracy			0.91	7365
macro avg	0.91	0.90	0.90	7365
weighted avg	0.91	0.91	0.91	7365



In [62]: # PRUEBA PRELIMINAR XGBOOST

```
# Eliminamos mas columnas
columnas_a_eliminar = ['stop_time_min', 'traj_time_mean', 'traj_mean_acc_mean',
                       'stop_time_max', 'traj_percent_2_2dec5_mean', 'stop_dist_mean',
                       'traj_percent_7_8_mean', 'traj_percent_1_2_mean',
                       'stop_time_mean', 'traj_percent_1_1dec5_mean', 'month_max_speed',
                       'traj_dist_min', 'to_starboard', 'stop_dist_max', 'traj_percent_0_0dec5_mean',
                       'traj_percent_8_9_mean', 'traj_percent_8_10_mean', 'traj_mean_speed_mean',
                       'N_stops', 'month_dist', 'traj_percent_10_12_mean', 'traj_percent_5_6_mean',
                       'traj_percent_6_8_mean', 'traj_percent_1dec5_2_mean',
                       'traj_percent_5_10_mean', 'traj_percent_0_2_mean', 'traj_percent_9_10_mean',
                       'traj_percent_4_6_mean', 'traj_percent_2_3_mean', 'traj_dist_mean',
                       'traj_max_speed_max', 'traj_percent_4_5_mean', 'N_Trajectories',
                       'month_mean_speed', 'traj_percent_3_4_mean', 'traj_percent_3_3dec5_mean',
                       'traj_percent_6_7_mean', 'traj_percent_0_1_mean',
                       'traj_percent_0dec5_1_mean', 'traj_percent_4_4dec5_mean']
```

```

# Eliminar las columnas especificadas, ignorando errores si alguna no existe
df_global_parameters_2_XG= df_global_parameters_1.drop(columns=columnas_a_eliminar, errors='ignore')

In [63]: df_global_parameters_2_XG.columns

Out[63]: Index(['shiptype', 'month_max_lat', 'month_min_lat', 'month_mean_lat',
               'month_max_lon', 'month_min_lon', 'month_mean_lon', 'month_max_diff_lat',
               'month_max_diff_lon', 'traj_dist_max', 'traj_mean_dec_mean',
               'stop_dist_min', 'traj_percent_2dec5_3_mean',
               'traj_percent_3dec5_4_mean', 'traj_percent_4dec5_5_mean',
               'traj_percent_2_4_mean', 'traj_percent_12_14_mean',
               'traj_percent_14_16_mean', 'traj_percent_16_18_mean',
               'traj_percent_18_20_mean', 'traj_percent_0_5_mean',
               'traj_percent_10_15_mean', 'traj_percent_15_20_mean',
               'traj_percent_over20_mean', 'draught', 'to_bow', 'to_stern', 'to_port',
               'len', 'wid', 'ldivw', 'ldivd', 'wdivd', 'area', 'grith', 'aml', 'amt',
               'vs', 'aol'],
              dtype='object')

In [64]: # PRIMERA PRUEBA PRELIMINAR XGBOOST
# eliminamos mas columnas
columnas_a_eliminar = ['N_Trajectories', 'traj_percent_3_3dec5_mean',
                       'traj_percent_6_7_mean', 'traj_percent_0_1_mean',
                       'traj_percent_0dec5_1_mean', 'traj_percent_4_4dec5_mean']

# Eliminar las columnas especificadas, ignorando errores si alguna no existe
df_global_parameters_2_XG= df_global_parameters_2_XG.drop(columns=columnas_a_eliminar, errors='ignore')

In [65]: # PARÁMETROS PRIMERA PRUEBA PRELIMINAR XGBOOST

# Asegurar que shiptype sea numérico
encoder = LabelEncoder()
df_global_parameters_2_XG["shiptype"] = encoder.fit_transform(df_global_parameters_2_XG["shiptype"])

# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_2_XG, test_size=0.2, random_state=43,
                                     stratify=df_global_parameters_2_XG["shiptype"])

# Separar características y variable objetivo
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train["shiptype"]
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test["shiptype"]

# Creación del modelo con los datos correctos
x_train_dmatrix = xgb.DMatrix(x_train, label=y_train)
x_test_dmatrix = xgb.DMatrix(x_test, label=y_test)

# Definir parámetros
parametros = {
    "booster": "gbtree",
    "num_class": len(y_train.unique()),
    "max_depth": 10,
    "eta": 0.2,
    "objective": "multi:softmax",
    "nthread": 2
}

rondas = 650
modelo = xgb.train(parametros, x_train_dmatrix, rondas) # Entrenar el modelo
prediccion = modelo.predict(x_test_dmatrix) # Hacer predicciones

# Evaluar el modelo
accuracy = accuracy_score(y_test, prediccion)
print(f"Precisión del modelo: {accuracy:.4f}")

# Reporte de métricas
reporte = classification_report(y_test, prediccion, digits=2)
print("\nReporte de Clasificación:\n", reporte)

# Obtén la importancia de las características usando 'gain'
importance_dict = modelo.get_score(importance_type='gain')

# Convierte el diccionario a DataFrame
fi = pd.DataFrame(list(importance_dict.items()), columns=['att', 'importance'])
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)

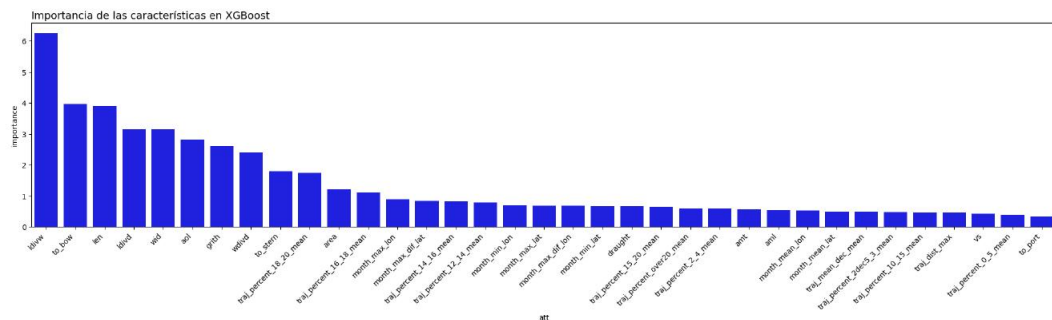
# Graficar la importancia usando seaborn
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(25, 5) # Ajustar el tamaño del gráfico
plt.xticks(rotation=45, ha='right') # Rotar etiquetas y alinearlas correctamente
plt.title("Importancia de las características en XGBoost", fontsize=14, loc="left") # Agregar título con "XGBoost" en cursiva
plt.show() # Mostrar el gráfico

Precisión del modelo: 0.9153

Reporte de Clasificación:

```

	precision	recall	f1-score	support
0	0.92	0.95	0.93	3650
1	0.92	0.92	0.92	886
2	0.87	0.87	0.87	466
3	0.90	0.84	0.87	1424
4	0.92	0.92	0.92	939
accuracy			0.92	7365
macro avg	0.91	0.90	0.90	7365
weighted avg	0.92	0.92	0.91	7365



```
In [37]: df_cinematic_parameters_3_XG = pd.read_csv('cinematic_parameters_3_jan_XG.csv')
df_cinematic_parameters_3_XG = df_cinematic_parameters_3_XG.drop(columns=['mmsi'])
```

```
In [38]: # EXPERIMENTO 3: XGBOOST - DINAMICOS SOLO

# Asegurar que shiptype sea numérico
encoder = LabelEncoder()
df_cinematic_parameters_3_XG["shiptype"] = encoder.fit_transform(df_cinematic_parameters_3_XG["shiptype"])

# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_cinematic_parameters_3_XG, test_size=0.2, random_state=43,
                                     stratify=df_cinematic_parameters_3_XG["shiptype"])

# Separar características y variable objetivo
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train["shiptype"]
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test["shiptype"]

# Creación del modelo con Los datos correctos
x_train_dmatrix = xgb.DMatrix(x_train, label=y_train)
x_test_dmatrix = xgb.DMatrix(x_test, label=y_test)

# Definir parámetros
parametros = {
    "boosted": "gbtree",
    "num_class": len(y_train.unique()),
    "max_depth": 10,
    "eta": 0.2,
    "objective": "multi:softmax",
    "nthread": 2
}

rondas = 650
modelo = xgb.train(parametros, x_train_dmatrix, rondas) # Entrenar el modelo
prediccion = modelo.predict(x_test_dmatrix) # Hacer predicciones

# Evaluar el modelo
accuracy = accuracy_score(y_test, prediccion)
print(f"Precisión del modelo: {accuracy:.4f}")

# Reporte de métricas
reporte = classification_report(y_test, prediccion, digits=2)
print("\nReporte de Clasificación:\n", reporte)

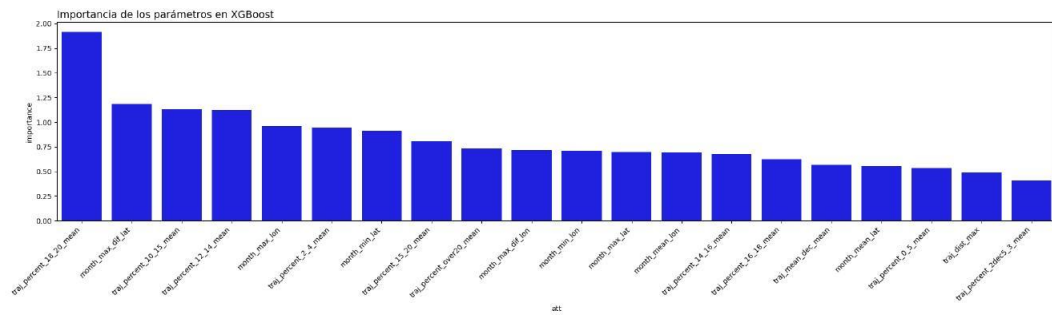
# Obtén la importancia de Las características usando 'gain'
importance_dict = modelo.get_score(importance_type='gain')

# Convierte el diccionario a DataFrame
fi = pd.DataFrame(list(importance_dict.items()), columns=['att', 'importance'])
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar la importancia usando seaborn
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(25, 5) # Ajustar el tamaño del gráfico
plt.xticks(rotation=45, ha='right') # Rotar etiquetas y alinearlas correctamente
plt.title("Importancia de los parámetros en XGBoost", fontsize=14, loc='left') # Agregar título
plt.show() # Mostrar el gráfico
```

Precisión del modelo: 0.7526

Reporte de Clasificación:		precision	recall	f1-score	support
	0	0.74	0.88	0.81	4003
	1	0.82	0.77	0.80	988
	2	0.78	0.59	0.67	527
	3	0.70	0.49	0.58	1597
	4	0.79	0.72	0.75	1057
	accuracy			0.75	8172
	macro avg	0.77	0.69	0.72	8172
	weighted avg	0.75	0.75	0.74	8172



```
In [43]: df_final_static = pd.read_csv("final_static_data_jan.csv")
df_cinematic_parameters_3_XG = pd.read_csv("cinematic_parameters_3_jan_XG.csv")
df_global_parameters_3_XG = pd.merge(df_cinematic_parameters_3_XG, df_final_static, left_on='mmsi', right_on='MMSI', how='inner')
df_global_parameters_3_XG = df_global_parameters_3_XG.drop(columns=['shiptype_y'])
df_global_parameters_3_XG = df_global_parameters_3_XG.rename(columns={'shiptype_x': 'shiptype'})
df_global_parameters_3_XG = df_global_parameters_3_XG.drop(columns=['mmsi', 'MMSI'])
```

```
In [46]: # EXPERIMENTO 4: XGBOOST - df_global_parameters_3_XG
# Asegurar que shiptype sea numérico
encoder = LabelEncoder()
df_global_parameters_3_XG["shiptype"] = encoder.fit_transform(df_global_parameters_3_XG["shiptype"])

# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_3_XG, test_size=0.2, random_state=43,
                                    stratify=df_global_parameters_3_XG["shiptype"])

# Separar características y variable objetivo
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train["shiptype"]
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test["shiptype"]

# Creación del modelo con los datos correctos
x_train_dmatrix = xgb.DMatrix(x_train, label=y_train)
x_test_dmatrix = xgb.DMatrix(x_test, label=y_test)

# Definir parámetros
parametros = {
    "booster": "gbtree",
    "num_class": len(y_train.unique()),
    "max_depth": 10,
    "eta": 0.2,
    "objective": "multi:softmax",
    "nthread": 2
}

rondas = 650
modelo = xgb.train(parametros, x_train_dmatrix, rondas) # Entrenar el modelo
prediccion = modelo.predict(x_test_dmatrix) # Hacer predicciones

# Evaluar el modelo
accuracy = accuracy_score(y_test, prediccion)
print(f"Precisión del modelo: {accuracy:.4f}")

# Reporte de métricas
reporte = classification_report(y_test, prediccion, digits=2)
print("\nReporte de Clasificación:\n", reporte)

# Obtén la importancia de las características usando 'gain'
importance_dict = modelo.get_score(importance_type='gain')

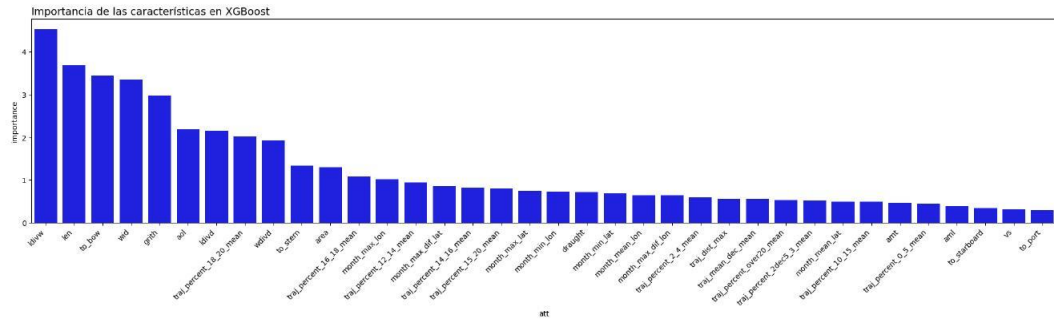
# Convierte el diccionario a DataFrame
fi = pd.DataFrame(list(importance_dict.items()), columns=['att', 'importance'])
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar la importancia usando seaborn
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(25, 5) # Ajustar el tamaño del gráfico
plt.xticks(rotation=45, ha='right') # Rotar etiquetas y alinearlas correctamente
plt.title("Importancia de las características en XGBoost", fontsize=14, loc='left') # Agregar título con "XGBoost" en cursiva
plt.show() # Mostrar el gráfico
```

Precisión del modelo: 0.9119

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.93	0.94	0.93	4003
1	0.92	0.93	0.92	988
2	0.85	0.85	0.85	527
3	0.89	0.85	0.87	1597
4	0.91	0.92	0.91	1057
accuracy			0.91	8172
macro avg	0.90	0.90	0.90	8172
weighted avg	0.91	0.91	0.91	8172



```
In [ ]: ## EXPERIMENTO 5: DATOS OBTENIDOS MEDIANTE LA OPTIMIZACION DEL ACCESO A LOS DATOS CARGANDOLOS EN UN DF

In [58]: parametros_cinematicos = pd.read_csv('parametros_cinematicos.csv')
parametros_cinematicos = parametros_cinematicos.drop(columns=['mmsi'])
parametros_cinematicos.reset_index(inplace=True)
parametros_cinematicos=parametros_cinematicos.drop('Unnamed: 0',axis=1)
parametros_cinematicos=parametros_cinematicos.drop('index',axis=1)

In [41]: # Experimento 5: RANDOM FOREST - DINAMICOS

# División en entrenamiento y prueba
df_train, df_test = train_test_split(parametros_cinematicos, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline (CPU y scikit-Learn)
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47, n_jobs=-1)) # n_jobs=-1 usa todos los núcleos de la CPU
])

# Entrenar el modelo en CPU
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo Random Forest: {accuracy:.4f}")

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=0))

# Obtener el modelo RandomForest dentro del Pipeline
forest = model.named_steps['classifier']

# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

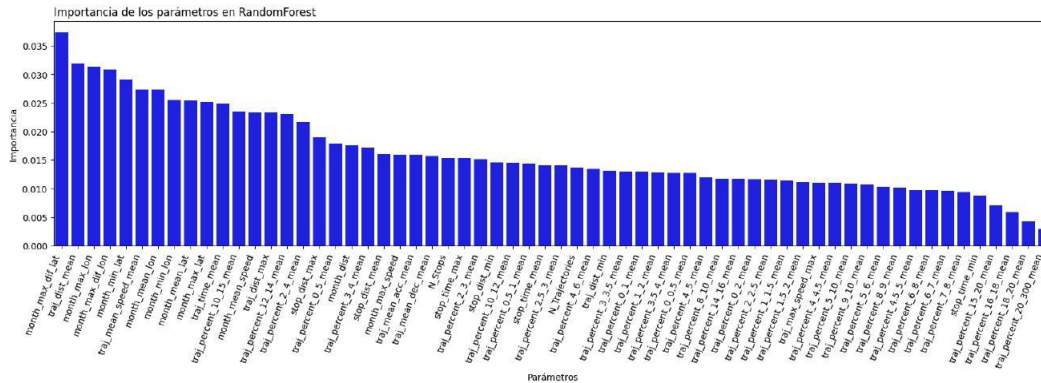
# Crear un DataFrame con la importancia de cada variable
fi = pd.DataFrame({
    'att': x_train.columns,
    'importance': importances
}).sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=fi, x='att', y='importance', color='b')

plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Parámetros")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest", loc="left")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()
```

Precisión del modelo Random Forest: 0.7648

	precision	recall	f1-score	support
0	0.75	0.92	0.82	4827
1	0.75	0.38	0.50	1869
2	0.78	0.79	0.78	1236
3	0.84	0.82	0.83	1100
4	0.81	0.61	0.70	577
accuracy			0.76	9609
macro avg	0.78	0.70	0.73	9609
weighted avg	0.77	0.76	0.75	9609



```
[59]: # EXPERIMENTO 5: XGBOOST DINAMICOS
# Asegurar que shiptype sea numérico
encoder = LabelEncoder()
parametros_cinematicos["shiptype"] = encoder.fit_transform(parametros_cinematicos["shiptype"])

# División en entrenamiento y prueba
df_train, df_test = train_test_split(parametros_cinematicos, test_size=0.2, random_state=43,
                                     stratify=parametros_cinematicos["shiptype"])

# Separar características y variable objetivo
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train["shiptype"]
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test["shiptype"]

# Creación del modelo con los datos correctos
x_train_dmatrix = xgb.DMatrix(x_train, label=y_train)
x_test_dmatrix = xgb.DMatrix(x_test, label=y_test)

# Definir parámetros
parametros = {
    "booster": "gbtree",
    "num_class": len(y_train.unique()),
    "max_depth": 10,
    "eta": 0.2,
    "objective": "multi:softmax",
    "nthread": 2
}

rondas = 650
modelo = xgb.train(parametros, x_train_dmatrix, rondas) # Entrenar el modelo
prediccion = modelo.predict(x_test_dmatrix) # Hacer predicciones

# Evaluar el modelo
accuracy = accuracy_score(y_test, prediccion)
print(f"Precisión del modelo: {accuracy:.4f}")

# Reporte de métricas
reporte = classification_report(y_test, prediccion, digits=2)
print("\nReporte de Clasificación:\n", reporte)

# Obtén la importancia de las características usando 'gain'
importance_dict = modelo.get_score(importance_type='gain')

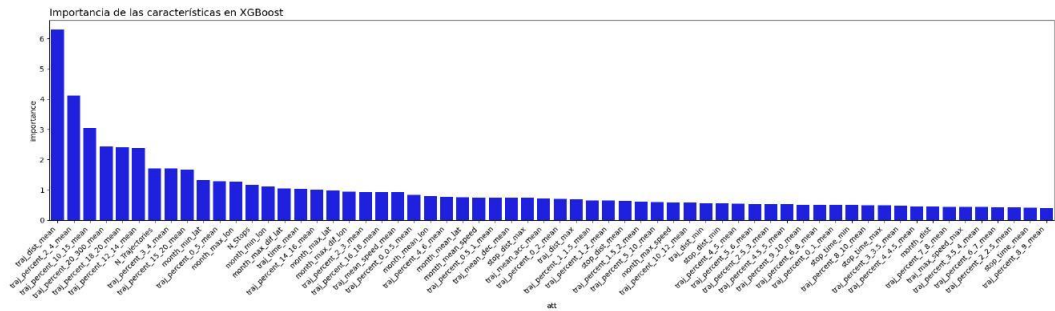
# Convierte el diccionario a DataFrame
fi = pd.DataFrame(list(importance_dict.items()), columns=['att', 'importance'])
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar la importancia usando seaborn
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(25, 5) # Ajustar el tamaño del gráfico
plt.xticks(rotation=45, ha='right') # Rotar etiquetas y alinearlas correctamente
plt.title("Importancia de las características en XGBoost", fontsize=14, loc='left') # Agregar título con "XGBoost" en cursiva
plt.show() # Mostrar el gráfico
```

Precisión del modelo: 0.8087

Reporte de Clasificación:		recall	f1-score	support
	precision			
0	0.80	0.90	0.85	3815
1	0.76	0.54	0.63	1502
2	0.81	0.82	0.82	983
3	0.88	0.88	0.88	907
4	0.83	0.72	0.77	481
accuracy			0.81	7688
macro avg		0.82	0.77	7688
weighted avg		0.81	0.81	7688

MEJORA DE LA EFICIENCIA COMPUTACIONAL DE UN SISTEMA DE PREDICCIÓN DE TIPO DE BUQUE UTILIZANDO PARÁMETROS DE TRAYECTORIAS Y TÉCNICAS DE IA



```
In [23]: df_final_static = pd.read_csv("final_static_data_jan.csv")
parametros_cinematicos = pd.read_csv('parametros_cinematicos.csv')
parametros_cinematicos.reset_index(inplace=True)
parametros_cinematicos=parametros_cinematicos.drop('Unnamed: 0',axis=1)
parametros_cinematicos=parametros_cinematicos.drop('index',axis=1)
df_global_parameters_final = pd.merge(parametros_cinematicos, df_final_static, left_on='mmsi', right_on='MMSI', how='inner')
df_global_parameters_final = df_global_parameters_final.drop(columns=['shiptype_y'])
df_global_parameters_final = df_global_parameters_final.rename(columns={'shiptype_x': 'shiptype'})
df_global_parameters_final = df_global_parameters_final.drop(columns=['mmsi', 'MMSI'])
```

```
In [24]: parametros_cinematicos
```

```
Out[24]:
```

	mmsi	shiptype	N_Trajectories	month_max_lat	month_min_lat	month_mean_lat	month_max_lon	month_min_lon	month_mean_lon	month_max_d
0	5	2	7	9.617068	-21.800053	8.195233	124.759834	-173.696609	120.697791	31.41
1	6	2	5	9.016103	-21.934406	8.016440	124.759880	-173.472229	119.047781	30.95
2	2829	0	8	-25.544207	-29.230938	-27.035603	-57.565269	-59.438766	-58.463124	3.66
3	3247	4	25	12.593527	11.925053	12.568194	121.948708	121.531662	121.545484	0.66
4	8086	4	27	9.887363	9.810703	9.819866	125.483872	125.299957	125.450000	0.07
...	...	...	...	...	...	...	...	...	...	...
38431	827002226	2	1	13.075975	13.047183	13.056416	100.905853	100.847847	100.888462	0.02
38432	848290500	0	1	14.625908	14.623703	14.624117	120.875465	120.873306	120.874869	0.00
38433	900004001	0	18	11.009773	10.313645	10.355707	124.597679	123.939156	123.981690	0.65
38434	924467387	3	0	51.516239	51.484604	51.499140	4.094505	4.049852	4.055688	0.03
38435	985524411	2	0	-8.772572	-8.772663	-8.772596	13.253562	13.253447	13.253446	0.00

38436 rows x 64 columns

```
In [40]: # Experimento 6: RANDOM FOREST - df_global_parameters_final

# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_final, random_state=47)

x_train = df_train.drop(columns=['shiptype'])
y_train = df_train['shiptype']
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test['shiptype']

# Creación del modelo con Pipeline (CPU y scikit-Learn)
pipe = Pipeline([
    ('scaler', MinMaxScaler()), # Normalización de datos
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=47, n_jobs=-1)) # n_jobs=-1 usa todos los núcleos de la CPU
])

# Entrenar el modelo en CPU
model = pipe.fit(x_train, y_train)

# Hacer predicciones
y_pred = model.predict(x_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo Random Forest: {accuracy:.4f}")

# Mostrar resultados
print(classification_report(y_test, y_pred, zero_division=0))

# Obtener el modelo RandomForest dentro del Pipeline
forest = model.named_steps['classifier']

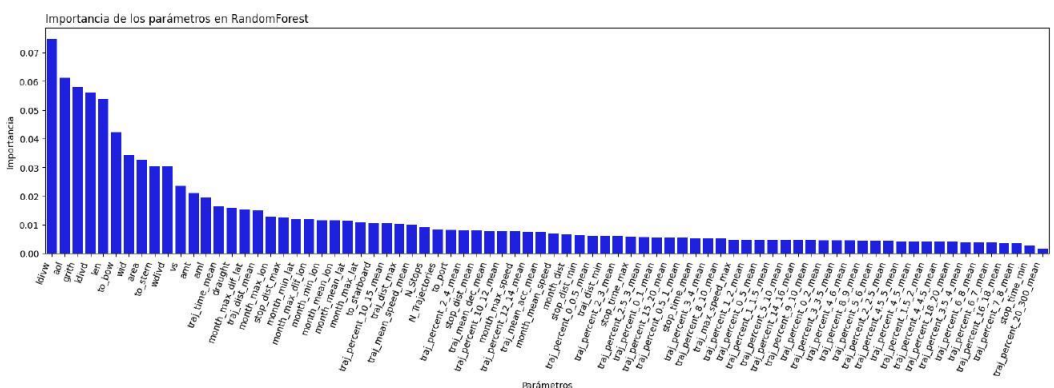
# Obtener importancia de características
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# Crear un DataFrame con la importancia de cada variable
fi = pd.DataFrame({
    'att': x_train.columns,
    'importance': importances
}).sort_values(by='importance', ascending=False).reset_index(drop=True)
```

```
# Graficar importancias
plt.figure(figsize=(20, 6))
sns.barplot(data=f1, x='att', y='importance', color='b')

plt.xticks(rotation=70, ha='right') # Rotar etiquetas 70° y alinearlas a la derecha
plt.xlabel("Parámetros")
plt.ylabel("Importancia")
plt.title("Importancia de los parámetros en RandomForest", loc="left")
plt.subplots_adjust(bottom=0.3) # Agregar margen inferior
plt.show()
```

Precisión del modelo Random Forest: 0.927				
	precision	recall	f1-score	support
0	0.89	0.96	0.93	4591
1	0.94	0.76	0.84	1776
2	0.91	0.91	0.91	1180
3	0.89	0.93	0.91	1180
4	0.88	0.84	0.86	595
accuracy			0.90	9242
macro avg	0.90	0.88	0.89	9242
weighted avg	0.90	0.90	0.90	9242



```
In [35]: # EXPERIMENTO 6: XGBOOST - df_global_parameters_final
```

```
# Asignar que shiptye sea número
encoder = LabelEncoder()
df_global_parameters_final["shiptye"] = encoder.fit_transform(df_global_parameters_final["shiptye"])

# División en entrenamiento y prueba
df_train, df_test = train_test_split(df_global_parameters_final, test_size=0.2, random_state=43,
                                     stratify=df_global_parameters_final["shiptye"])

# Separar características y variable objetivo
x_train = df_train.drop(columns=['shiptye'])
y_train = df_train["shiptye"]
x_test = df_test.drop(columns=['shiptye'])
y_test = df_test["shiptye"]

# Creación del modelo con los datos correctos
x_train_dmatrix = xgb.DMatrix(x_train, label=y_train)
x_test_dmatrix = xgb.DMatrix(x_test, label=y_test)

# Definir parámetros
parametros = {
    "booster": "gbtree",
    "num_class": len(y_train.unique()),
    "max_depth": 10,
    "eta": 0.2,
    "objective": "multi:softmax",
    "nthread": 2
}

rondas = 650
modelo = xgb.train(parametros, x_train_dmatrix, rondas) # Entrenar el modelo
prediccion = modelo.predict(x_test_dmatrix) # Hacer predicciones

# Evaluar el modelo
accuracy = accuracy_score(y_test, prediccion)
print(f"Precisión del modelo: {accuracy:.4f}")

# Reporte de métricas
reporte = classification_report(y_test, prediccion, digits=2)
print("\nReporte de Clasificación:\n", reporte)

# Obtén la importancia de las características usando 'gain'
importance_dict = modelo.get_score(importance_type='gain')

# Convierte el diccionario a DataFrame
fi = pd.DataFrame(list(importance_dict.items()), columns=['att', 'importance'])
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)

# Graficar la importancia usando seaborn
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(25, 5) # Ajustar el tamaño del gráfico
```

