UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FCEN
FACULTAD DE CIENCIAS
EXACTAS Y NATURALES

Técnicas de Aprendizaje Automático Aplicadas a Simulaciones Numéricas de Colisiones de Material Granular Poroso

Seminario de investigación y/o desarrollo

Universidad Nacional de Cuyo

Facultad de Ciencias Exactas y Naturales

Licenciatura en Ciencias Básicas con Orientación en Física

Alumno: Daniela N. Rim

Director: Dr. Luis G. Moyano

Co-director: Dr. Emmanuel N. Millán

Supervisores:

Dr. Carlos J. Ruestes

Dr. David Monge

Agradezco y dedico este trabajo a mi Padre Celestial. Esto ha sido posible sólo por Él.

AGRADECIMIENTOS

Agradezco el incondicional amor y apoyo de mis padres: mi papá Daniel, su constante preocupación y cuidado, siempre alentándome y creyendo en mí. A mi mamá Sandra, consejera y modelo de mujer. A mis abuelas Margarita y Mi-Suk, a mi tía Ame. A mis hermanas Este, Mary y Ruth, fuentes constantes de aliento. A la familia espiritual, presentes desde el principio.

No podría imaginar dos directores más perfectos que Luis y Emmanuel. Si este trabajo existe hoy en tiempo y forma es gracias a ellos: son un verdadero modelo de excelencia, tanto en lo profesional como en lo humano. Estoy profundamente agradecida por ellos.

Agradezco a los jurados Carlos y David, por la buena voluntad de realizar las correcciones en tiempo tan acotado.

Agradezco al profesor Pablo Kaluza, quien me introdujo en el ámbito de la investigación, acompañando todo el camino de cerca y de lejos. Agradezco los consejos y el respaldo del Dr. Eduardo Bringa, la buena disposición en todo momento de Belén Planes (de quien nació la idea de este trabajo), a todos los excelentes profesores de esta carrera cuyas enseñanzas, directa o indirectamente, aspiré a plasmar en estas hojas.

A los mejores amigos que existen: Caro, Clarita y Johnny, han estado siempre. A Gime, Sofi, Santy, Guille, Pony y Flor.

Gin[1] *a body meet a body*

Flyin' through the air.

Gin a body hit a body,

Will it fly? And where?

Ilka[2] *impact has its measure,*

Ne'er[3] *a ane*[4] *hae*[5] *I,*

Yet a[6] *the lads they measure me,*

Or, at least, they try.

"Rigid Body Sings"[7]

James Clerk Maxwell

[1]If [2]Every [3]Never [4]one [5]have [6]all [7]Fragmento de la parodia realizada por el físico James C. Maxwell (1831-1879) del poema/canción popular "Comin' Through the Rye" por Robert Burns

ÍNDICE TEMÁTICO

Página

Página

SÍMBOLOS

$\mathcal{D}$	Distribución de probabilidad generadora de $\mathcal{X}$.
$\mathcal{X}$	Conjunto de dominio.
$\mathcal{Y}$	Conjunto de etiquetas.
f	Función de etiquetado correcto.
$\mathcal{S}$	Conjunto de entrenamiento.
h	Regla de predicción, predictor, hipótesis o clasificador.
$L_{(\mathcal{D},f)}$	Error verdadero o de generalización.
$L_S(h)$	Error de entrenamiento o error empírico.
$\mathcal{H}$	Clase de hipótesis.
$<,>$	Producto punto o producto interno.
ϕ	Factor de llenado.
v_0	Velocidad inicial de proyectil.
T_f	Tiempo total empleado en correr una simulación hasta el final.
T_c	Tiempo de corte optimizado.
G_t	Porcentaje de ganancia temporal.
R_g	Radio granular.

ABREVIACIONES

MD	Dinámica Molecular.
AA	Aprendizaje Automático.
A	Algoritmo de aprendizaje.
NFL	Teorema "No Free Lunch".
SVM	*Support Vector Machines* o Máquinas de Vectores de Soporte.
$OCSVM$	*One Class svm* o Máquinas de Vectores de Soporte de Una Clase.
RF	*Random Forest* o Bosques Aleatorios.
MC	Matriz de Confusión.
VP	Verdadero/s Positivo/s.
VN	Verdadero/s Negativo/s.
FP	Falso/s Positivo/s.
FN	Falso/s Negativo/s.

RESUMEN

En el área de la física, las aplicaciones de nuevas herramientas computacionales ha permitido un gran avance en las investigaciones científicas, y este es el caso de la física de materiales, en donde las simulaciones computacionales han tenido un papel central desde hace décadas, siendo hoy una parte integral del área. Esto no solamente ha provisto los medios para realizar complejos experimentos científicos, sino también ha permitido obtener resultados en forma cada vez más eficiente.

En este seminario se presenta un método computacional para obtener información sobre el desenlace de simulaciones de colisiones de granos porosos antes de que las mismas finalicen. Utilizando un sistema de aprendizaje automático se busca ahorrar tiempo real de simulación mediante la implementación de algoritmos supervisados, los cuales "aprenden" a categorizar las partículas constituyentes de una simulación. Se propuso primeramente etiquetar ciertos datos de entrenamiento de forma automática, haciendo uso de algoritmos de aprendizaje no supervisado más una regla heurística para definir un tiempo de finalización. Seguidamente, se entrenó a los algoritmos de aprendizaje supervisado mediante estos datos etiquetados y se evaluó su capacidad de generalizar las predicciones obtenidas del aprendizaje al categorizar el desenlace de las colisiones en nuevas simulaciones.

Con la metodología aquí propuesta se logra predecir el desenlace de 35 simulaciones de distintos parámetros generales y tamaños (con un 95 % de aciertos en cada una) ahorrando un 73.3 % del tiempo total que tomaría ejecutarlas hasta el tiempo de finalización.

1. INTRODUCCIÓN

El desarrollo de nuevas tecnologías y el surgimiento de una vasta cantidad de herramientas computacionales en estas últimas décadas han permitido la exploración de áreas del conocimiento nunca antes accedidas. En especial, las áreas científicas se han visto altamente beneficiadas por la implementación de estas técnicas. Gracias a los experimentos computacionales, se han podido reforzar teorías y experimentos, obtener resultados de sistemas impracticables en un laboratorio, y abrir nuevas líneas de investigación sin precedentes, entre muchas otras cosas. Además, ha sido posible el procesamiento de datos científicos mediante programas computacionales de visualización de datos, generadores de gráficos, de análisis estadísticos por medio de lenguajes de programación, entre otros ejemplos. Esto permite no solamente que la herramienta computacional provea los medios para realizar un experimento científico, sino también un análisis y posible mejora de los resultados obtenidos.

Un ejemplo de este tipo de herramientas es el Aprendizaje Automático, un área en las Ciencias de la Computación con diversas aplicaciones, entre ellas el área de la física de materiales, y específicamente en el contexto de simulaciones numéricas. El éxito del Aprendizaje automático no sólo se refleja en recientes publicaciones científicas con aplicaciones concretas [1–3], sino también en el surgimiento de distintas iniciativas en varios lugares del mundo, de carácter institucional o privado, centradas en la utilización de este tipo de algoritmos para acelerar y mejorar el estudio de materiales[1] [2] [3] [4] [4].

El presente seminario presenta resultados situados precisamente en la intersección del Aprendizaje Automático y de una parte del ámbito de la física de los materiales: se

[1] https://www.mgi.gov/ [2] https://citrine.io/research/open-citrination-platform/
[3] https://nomad-coe.eu/ [4] https://www.nature.com/articles/d41586-019-00885-5

busca mejorar la eficiencia de simulaciones numéricas de física de materiales mediante algoritmos de aprendizaje.

1.1 Antecedentes de la propuesta física

En las últimas décadas, el estudio teórico de la física de materiales se ha visto beneficiado gracias a la realización de simulaciones computacionales de sistemas físicos. Por ejemplo, se han estudiado impactos de proyectiles sobre materiales a diversas escalas y tamaños, siendo factible simular no sólo cuerpos grandes (escalas centimétricas) tratados como un continuo, sino también interacciones de cada una de las partículas constituyentes de los cuerpos simulados (escalas micro y sub-micrométricas).

La posibilidad de simular colisiones mediante una computadora permitió el estudio de diversos materiales de impactos, como por ejemplo blancos granulares donde los tamaños de granos son lo suficientemente grandes para despreciar las fuerzas cohesivas entre ellos [5–8]. También ha sido factible simular impactos de materiales cohesivos al tomar en cuenta la complejidad de las interacciones físicas entre los granos que lo constituyen, principalmente gobernadas por fuerzas de cohesión (Van der Waals), fuerzas elásticas y fuerzas de fricción (de deslizamiento, de rodadura y de torsión). Estas interacciones han sido formuladas y teorizadas por Dominik y Tielens [9].

Con la mira en aplicaciones de astrofísica, se publicaron diversos trabajos de códigos computacionales para simulaciones de materiales porosos cohesivos teniendo en cuenta los detalles de interacción mencionados, por ejemplo el de Paszun y Dominik [10] o Wada et al. [11–14], entre otros. La desventaja de tener en cuenta todas las interacciones entre los granos constituyentes es la alta demanda computacional de cálculo, y por ende estos trabajos estuvieron limitados a sistemas de unos pocos miles de ellos, lejos de situaciones realistas.

Un código computacional más eficiente y con un modelo mejorado de fricción fue propuesto por Ringl et al. [15], quienes utilizando el software LAMMPS[5], lograron

[5] `https://lammps.sandia.gov/`

simular del orden de $\sim 10^6$ partículas en un solo procesador con las consideraciones físicas básicas. Para modelar los clusters, emplearon una estrategia de simulación ya ampliamente utilizada para simular colisiones granulares: Dinámica Molecular (MD). La base de MD es resolver las ecuaciones de movimiento de Newton para un conjunto de partículas que interactúan a través de un campo de fuerza. Este método tiene la ventaja de permitir incorporar las propiedades complejas de las interacciones del polvo. Gracias a este trabajo y a la publicación del código utilizado por los autores, se facilitó el estudio y la exploración de nuevas configuraciones de colisiones de materiales porosos.

El equipo de investigación de Astrofísica e Impactos de Clusters de *SiMAF* (Simulaciones en Materiales de Astrofísica y Física) radicado en la ciudad de Mendoza, Argentina, dependiente de la Universidad de Mendoza, ha tomado esta línea de investigación, también colaborando con los autores del trabajo citado [15]. Las investigaciones se concentran en el estudio de impactos sobre materiales granulares o de partículas de polvo (también llamados granos, *clusters* o aglomerados) formados por granos más pequeños de sílica o SiO_2, a escalas micrométricas. En astrofísica, comprender el mecanismo físico subyacente en estas escalas resulta ser fundamental cuando, por ejemplo, se trata de explicar la formación nuevos cuerpos planetarios. La colisión entre partículas de polvo deviene en un crecimiento por aglomeración entre las partículas o en una fragmentación de las mismas. En los anillos protoplanetarios, el aumento de tamaño de granos mediante agregación de partículas colisionantes sugiere a este fenómeno como uno de los pasos iniciales de la formación de planetas [16, 17].

El equipo de *SiMAF* ha realizado estudios en impactos de esferas sólidas sobre lechos granulares porosos (formación de cráteres), eyección de material [18], y fenómenos de compactación y fragmentación de granos porosos del mismo tamaño con distintos parámetros de impacto [19]. Además de estudios estructurales también se analizaron las consecuencias sobre la formación de cráteres en lechos granulares porosos al variar parámetros generales como la velocidad inicial y tamaño del proyectil colisionante, siendo este también de un material granular [20].

En trabajos como el citado [19] se han analizado colisiones entre granos de misma masa (simétricos). Sin embargo, en una situación más realista, las colisiones son asimétricas en el sentido de poseer los aglomerados distinta masa y tamaño [21]. Los modelos actuales estiman la tasa de fragmentación crítica de granos con estas características en el orden de $1 - 10\frac{m}{s}$ [22–24], y en ninguno de estos trabajos se discute la dependencia del desenlace de la colisión (fragmentación o aglomeración) con respecto a la porosidad de los colisionantes.

Recientemente el equipo de *SiMAF* ha presentado un trabajo a publicar [25] en donde se exploran colisiones asimétricas con velocidades de proyectil pequeñas, enfatizando el rol de la porosidad de los granos. Se han encontrado fragmentaciones a velocidades menores a las estimadas en [22–24] y una dependencia de este fenómeno con la porosidad de los aglomerados. Tales resultados pueden implicar modificaciones en los modelos de aglomeración/fragmentación utilizados hasta ahora. El trabajo presente se enmarca dentro de estas investigaciones, analizando simulaciones de colisiones del tipo tratado en [25], con el objeto de mejorar el uso de los recursos computacionales de manera de potenciar la capacidad de las simulaciones numéricas propuestas.

1.2 Motivación

Las herramientas y técnicas computacionales actuales permiten simular distintas configuraciones de granos, conformados a su vez por grandes cantidades de partículas. Se mencionó anteriormente que a mayor cantidad de partículas constituyentes, más cerca estará la simulación de representar una colisión real. Sin embargo, cuanto más realista una simulación, mayor es el costo computacional de las simulaciones de MD en términos del *hardware* necesario para ejecutarlas y el tiempo de cómputo de los mismos.

Parte del costo temporal que las simulaciones suponen se ha resuelto mediante la Computación de Alto Rendimiento o *HPC* por sus siglas en inglés, la cual permite resolver problemas computacionalmente costosos utilizando algoritmos paralelos y

clusters de computadoras o clusters. Existen múltiples formas de implementación de *hardware* y *software* de *HPC*, como por ejemplo, supercomputadoras de miles de núcleos CPU, o en los últimos 10 años, la utilización de Unidades de Procesamiento Gráfico o *GPUs*. Generalmente, el acceso a supercomputadoras está restringido a un grupo selecto, por ello se ha vuelto común en el ámbito de investigación instalar un pequeño cluster de computadoras o usar GPUs como parte del equipo de trabajo (o una combinación de ambos) [26].

Los recursos computacionales de los cuales disponga el investigador imponen ciertas restricciones a las escalas temporal y dimensional de las simulaciones. La escala temporal se refiere al tiempo en el cual transcurren los fenómenos simulados y la dimensional al tamaño de la muestra simulada. El aumento de ambas implica un aumento de tiempo de cómputo, que puede ser disminuido hasta cierto punto paralelizando; descomponer el dominio de la muestra y usar múltiples núcleos simultáneamente que resuelvan cada uno un subdominio del sistema original. También el tiempo de cómputo puede disminuirse mediante procesadores más veloces. Sin embargo, la utilización de recursos como *GPU* o varios núcleos *CPU* tiene un límite, en el cual el tiempo de cómputo para una muestra determinada no puede disminuirse más, ya sea porque no se dispone de mejores equipos, o por las limitaciones mismas de ciertos parámetros de la simulación. Un ejemplo de este último sería dividir el sistema original en tantos subdominios que se pierde más tiempo comunicando información entre ellos que calculando las relaciones dentro de cada uno. Es por ello que aún usando estas estrategias, una sola simulación del tipo utilizado para [25] puede llegar a tardar 3 meses en realizarse.

Ante una limitación de equipos, la simplificación física de los sistemas no es una opción que se contemple para acortar el tiempo de cómputo; se desea mantener todas las interacciones que corresponden a partículas de este tamaño para tener una mayor exactitud en la simulación del comportamiento real. Se necesita otra herramienta capaz de disminuir el tiempo de cómputo de este tipos de simulaciones. ¿Podría alguna característica intrínseca de una simulación de colisión dar alguna pauta de

su forma final en una escala temporal menor a la programada? Toda la información sobre las partículas obtenidas de las configuraciones a distintos tiempos de simulación queda plasmada en forma de grandes cantidades de datos, imposibles de manipular manualmente para obtener información útil. Existe un área de las ciencias de la computación denominada Aprendizaje Automático (o *Machine Learning*) basada en el desarrollo de algoritmos informáticos para transformar datos en acción inteligente. El Aprendizaje Automático se ha convertido en una de las herramientas más populares y adecuadas para manipular un volumen de datos del orden de magnitud planteado aquí.

Mediante algoritmos de aprendizaje, se pueden encontrar relaciones, patrones, o características en los datos a partir de los cuales se realizan predicciones (generalizaciones) de los mismos.

El Aprendizaje Automático ha sido usado exitosamente en combinación con simulaciones de MD en trabajos previos, tales como: mejorar el reconocimiento de proteínas [27], predecir las energías de atomización en moléculas orgánicas [28], acelerar las simulaciones ab initio MD [29], entre otras. No se han encontrado publicaciones de aplicaciones de Aprendizaje Automático en colisiones de granos, probablemente debido a lo reciente del tema.

Por medio de este trabajo, se hace una contribución a la investigación sobre colisiones de granos porosos al buscar maneras de disminuir el tiempo de simulación mediante técnicas de Aprendizaje Automático, ante limitaciones impuestas por el *hardware* disponible.

1.3 Objetivos

Los objetivos propuestos por el presente Seminario de Investigación son los siguientes:

1. Analizar y explorar algoritmos de Aprendizaje Automático supervisados y no supervisados adecuados a la tarea de clasificación que desea llevarse a cabo, con el fin de disminuir el tiempo de simulación de colisiones.
2. Observar la influencia de los distintos parámetros generales de las colisiones sobre el desenlace de las mismas.
3. Analizar la aplicabilidad de los métodos de optimización propuestos entre distintos subespacios del espacio de parámetros de simulación.

1.4 Hipótesis propuestas y adelanto de resultados

Las hipótesis formuladas a partir de los objetivos se detallan a continuación:

- **Hipótesis 1:** Es posible diseñar una metodología de optimización temporal de simulaciones numéricas basada en algoritmos de Aprendizaje Automático para un conjunto dado de parámetros de simulación.
- **Hipótesis 2:** Es posible predecir el desenlace de una colisión mucho antes de su finalización teniendo en cuenta un conjunto específico de sus variables y parámetros.
- **Hipótesis 3:** Existe un subespacio del espacio de parámetros generales de las simulaciones con el cual se puede realizar un modelo de optimización temporal aplicable a simulaciones con otros parámetros generales, con resultados equivalentes.

Mediante estas propuestas se logró obtener información sobre el desenlace de la colisión de granos asimétricos de 35 simulaciones en un tiempo total de 16 días (con un 5 % de error), siendo dos meses el tiempo total empleado en ejecutar estas simulaciones hasta el desenlace de la colisión.

2. MARCO TEÓRICO: NOCIONES DE APRENDIZAJE AUTOMÁTICO

2.1 Introducción

Machine Learning o Aprendizaje Automático (AA) es un campo interdisciplinario de las ciencias de la computación cuyo fin, como dice su nombre, es programar a las computadoras para *aprender*. Comúnmente se entiende el aprendizaje como una forma de adquirir conocimiento de algo por medio de la experiencia. El aprendizaje humano es complejo y aún muchos de los mecanismos que subyacen a esta actividad permanecen vedados. Sin embargo, en AA no se pretende en una computadora un proceso de aprendizaje semejante al humano (característica que lo diferencia de la rama de inteligencia artificial), sino el desarrollo de una herramienta para complementar la realización de tareas complejas no factibles para la capacidad humana, aprovechando las ventajas que ofrece la computación.

El punto de partida para armar un sistema de AA es tener una tarea a realizar. Muchos tipos de tareas pueden resolverse con AA, se nombran algunas a continuación:

- **Clasificación:** Se necesita especificar a cuál de una serie de categorías pertenece un objeto dado. Por ejemplo, en este trabajo, la tarea a realizar es una clasificación: etiquetar partículas según pertenezcan a cual fragmento luego de una colisión.
- **Regresión:** se desea la predicción de un valor numérico dados ciertos datos. Es una de las tareas más comunes.
- **Detecciones de anomalías:** se espera que, al observar en detalle un conjunto de eventos u objetos, se detecten los casos inusuales o atípicos.
- **Síntesis y muestreo:** se desea generar ejemplos nuevos que sean similares a los que ya se tienen.

- **Atribución de valores faltantes:** cuando se desea completar un conjunto con valores ausentes.
- **Eliminar ruidos:** se desea predecir un caso correcto a partir de una versión dañada del mismo por algún proceso desconocido.

Los "alumnos" de AA que deben aprender a realizar la tarea son los algoritmos. La primer y más sencilla forma de aprender es mediante la memoria: por la experiencia pasada se pueden reconocer y categorizar entidades. El siguiente paso es lograr una *inferencia inductiva*: reconocer entidades no conocidas con anterioridad, es decir, *generalizar a partir de ejemplos individuales.* Este es el objetivo de aprendizaje principal para un algoritmo.

Es sabido que la inferencia inductiva puede llevar a conclusiones falsas, y la mente humana posee sentido común para discriminar estas conclusiones de las que tienen sentido. Sin embargo, la ausencia intrínseca de esta cualidad en las máquinas requiere encontrar y especificar a los algoritmos los principios para eludir conclusiones ilógicas.

En un sistema básico de AA para el tipo de tarea de clasificación que desea llevarse a cabo en este trabajo en particular se debe proveer al **algoritmo** (llamado A de ahora en adelante) de AA una **Entrada**, que representa la experiencia, y la **Salida** será la acción o habilidad lograda mediante el aprendizaje. En las secciones siguientes se analizarán con más detalle la relación entre estos tres elementos. Utilizaremos la notación del libro de Shalev-Schwartz, 2014 [30].

2.2 El entorno de A: Entrada y Salida

2.2.1 La Entrada

Sea $\mathcal{D}$ una distribución de probabilidad generadora de un conjunto arbitrario $\mathcal{X} = \{x_i\}$, con $i \in 0, ..N$. El conjunto $\mathcal{X}$ se denomina conjunto de dominio y es la colección de objetos que se desean clasificar (también llamados ejemplos). Por ejemplo, en nuestro trabajo, $\mathcal{X}$ es el conjunto de N partículas que conforman las estructuras

que colisionan. Cada elemento de $\mathcal{X}$ está representado por un *vector de características*, por ejemplo las posiciones y velocidades de la partícula i para un tiempo particular.

Para clasificar se necesita un conjunto de etiquetas $\mathcal{Y}$ siendo un caso común el binario, por ejemplo $\mathcal{Y} = \{1, 0\}$. Además, asumimos que existe una *función de etiquetado* "correcta" $f : \mathcal{X} \to \mathcal{Y}$ tal que $y_i = f(x_i)$ para todo i (para el caso binario, $y_i \in \{1, 0\}$).

Se muestrea cada uno de los elementos x_i de $\mathcal{X}$ distribuido de acuerdo a $\mathcal{D}$ y se los etiqueta mediante f para generar el *conjunto de entrenamiento* $\mathcal{S}$, definido como una secuencia finita de pares $\mathcal{X} \times \mathcal{Y}$: $\mathcal{S} = ((x_1, y_1)...(x_m, y_m))$, donde cada (x_i, y_i) representa un punto del dominio etiquetado.

Se debe tener en cuenta que A no tiene información directa de la distribución $\mathcal{D}$, como tampoco sabe cuál es la función f. De hecho, esto es lo que se quiere encontrar. $\mathcal{S}$ es la Entrada a la cual tiene acceso A y es lo único que puede conocer de $\mathcal{D}$ y f.

2.2.2 La Salida

Se quiere que A devuelva una *regla de predicción* o *predictor* (también llamado hipótesis o clasificador o modelo) $h : \mathcal{X} \to \mathcal{Y}$ que tenga la capacidad de predecir las etiquetas de nuevos elementos de dominio.

Definimos el *error verdadero o de generalización* de un clasificador h, $L_{(\mathcal{D},f)}$, como la probabilidad de tomar al azar un x generado por $\mathcal{D}$ tal que $h(x) \neq f(x)$. El error se mide respecto a $\mathcal{D}$ y a f.

Dado un $\mathcal{S}$, este error es el que determina la elección de h: no se quiere cualquier h_S, sino aquel cuyo error $L_{(\mathcal{D},f)}(h)$ sea el menor posible. La dificultad, como ya mencionamos, es que A no dispone ni de $\mathcal{D}$ ni de f. La alternativa es mirar lo que sí se tiene: el conjunto de entrenamiento $\mathcal{S}$. Definimos el *error de entrenamiento* (o error empírico) $L_S(h)$ al error que comete el clasificador h cuando actúa sobre la muestra de entrenamiento. Por ende, el objetivo pasa a ser encontrar un clasificador tal que el error de entrenamiento sea mínimo, es decir, que ese h tenga un buen desempeño.

Para tener un predictor que se desempeñe bien en el conjunto de entrenamiento $\mathcal{S}$ y también sobre toda la distribución subyacente de datos, se puede buscar el clasificador deseado en un espacio restringido denominado *clase de hipótesis* $\mathcal{H}$. La clase de hipótesis es un conjunto de predictores $\{h_k\}$ tal que $h_k : \mathcal{X} \to \mathcal{Y}$ y se busca entre estos aquel que de el mínimo error de $L_S(h)$. En definitiva, estamos sesgando a A al restringir su elección de predictor, y esta restricción debe realizarse, indudablemente, con algún conocimiento previo de la tarea que quiere realizarse.

En la Fig. 2.1, se esquematiza un sistema de AA con las nociones descritas en esta sección.

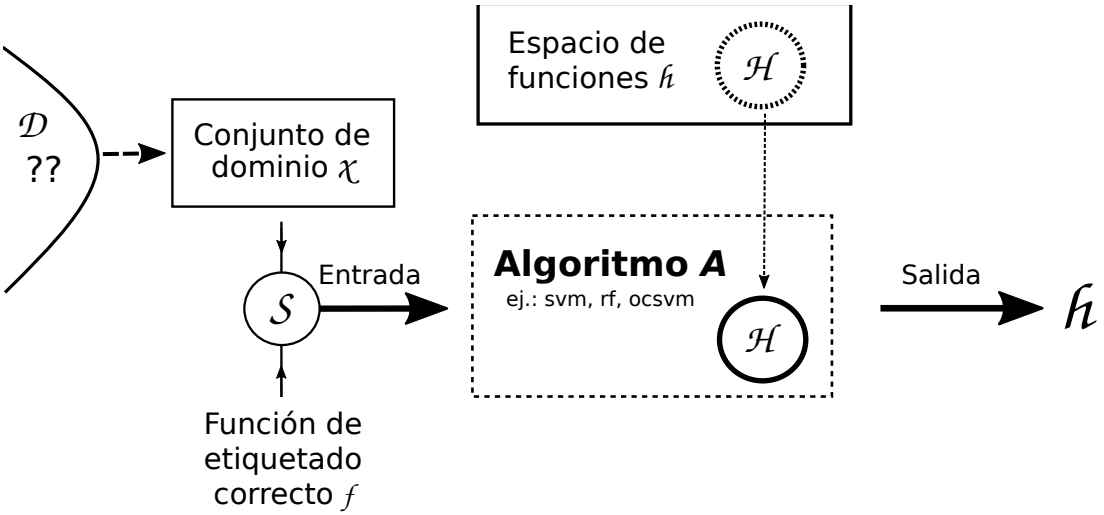

Figura 2.1.: Dada una clase de hipótesis $\mathcal{H}$, un sistema AA básico funciona como sigue: A recibe un conjunto de entrenamiento $\mathcal{S}$ proveniente de un conjunto de dominio $\mathcal{X}$ generado por una distribución de probabilidad desconocida $\mathcal{D}$ y etiquetado por una función $f : \mathcal{X} \to \mathcal{Y}$. A evalúa el error verdadero de cada $h \in \mathcal{H}$ y devuelve el miembro que minimice el error de entrenamiento.

Errores asociados a la utilización de $\mathcal{H}$

El sesgo de un $\mathcal{H}$ tiene ventajas, pero también introduce nuevas dificultades. Por ejemplo, ¿cómo sabemos cuál clase de hipótesis $\mathcal{H}$ de todas las posibles tendrá el conjunto de predictores con el menor error de entrenamiento? Además se quiere elegir el $\mathcal{H}$ que tenga el predictor h con el menor error. Cuanto mayor sea la clase de

hipótesis, mayor probabilidad de incluir esa h, pero agrandar el tamaño de la clase tiene su costo.

La elección de $\mathcal{H}$ tiene dos errores asociados:

- **Error de aproximación:** También llamado *variance* en la literatura en inglés, mide cuanto error se tiene por haber elegido un $\mathcal{H}$ específico. No depende del tamaño de la muestra, sino de $\mathcal{H}$: cuanto mayor es $\mathcal{H}$, menor es este error.
- **Error de estimación:** También llamado *bias* en la literatura en inglés, es la diferencia entre el error de aproximación y el error $L_S(h)$. Este error surge debido a que este error de entrenamiento es solo una estimación del error real, y por ende el predictor que minimiza el error de entrenamiento es solo una estimación del que minimiza el error real. Este error disminuye cuando el tamaño de $\mathcal{S}$ lo hace y aumenta cuando el tamaño de $\mathcal{H}$ aumenta (un mayor tamaño implica generalmente un aumento de complejidad).

Se quiere que ambos valores de error sean los menores posibles, la dificultad estriba en que la disminución de uno suele implicar el aumento del otro. En la sección siguiente se mostrará la importancia del tamaño de $\mathcal{H}$ para evitar dos inconvenientes de frecuente ocurrencia en tareas de AA: el sobreajuste y el subajuste.

Sobreajuste y subajuste

El tamaño y elección de $\mathcal{H}$ son dos factores cruciales para obtener un equilibrio entre los errores de aproximación y estimación. Si se tiene un $\mathcal{H}$ muy grande, el error de aproximación será pequeño, pero el de estimación será mayor, pues tiene más posibilidades de funcionar bien sólo para el conjunto de entrenamiento, y no para el resto de los datos generados por $\mathcal{D}$. A este fenómeno se le denomina *overfitting* o sobreajuste, y en forma simple es la incapacidad de la Salida de generalizar, es decir, clasificar correctamente casos no contenidos en $\mathcal{S}$. Está relacionado con una gran varianza; el algoritmo entrenará detalladamente con todos los valores provistos, aún con los anómalos.

Si $\mathcal{H}$ es muy pequeño, el error de estimación se reduce pero puede aumentar el error de aproximación, conduciendo a un *underfitting* o subajuste de los datos. Esto sucede cuando hay un gran *bias* o sesgo en la Salida h, es decir, existe una tendencia hacia un resultado específico por asunciones erróneas del algoritmo respecto de la estructura de $\mathcal{D}$ adquiridas durante su entrenamiento. Cuando se produce un subajuste, ni el entrenamiento del algoritmo ni su generalización resultan correctos, y esto es un indicador de que el algoritmo no es el indicado para realizar esta tarea.

Aquí entra en juego la *compensación de complejidad del sesgo*: deseamos encontrar un $\mathcal{H}$ lo suficientemente "completo" y seguir teniendo un error de estimación razonable.

2.3 El algoritmo A

Debido a la diversidad de las tareas a realizar, existen variados algoritmos de AA y la lista crece con el tiempo. Aunque no existe una clasificación formal y cerrada de los algoritmos, podemos distinguir dos grupos según el tipo de interacción que tienen con el entorno que los rodea: los algoritmos *supervisados* y *no supervisados*.

- **Algoritmos supervisados:** entendiendo el aprendizaje como un proceso para obtener habilidades mediante la experiencia, los algoritmos supervisados se mueven en un escenario en donde la *experiencia* (el conjunto de entrenamiento) posee información significativa, ausente en los ejemplos sobre los cuales se quiere aplicar su habilidad. La habilidad adquirida tiene como objetivo predecir la información que falta de los datos de evaluación. El entorno es un "profesor" que supervisa al alumno al darle información extra (datos etiquetados).
- **Algoritmos no supervisados:** no hay distinción entre el entrenamiento y los elementos a evaluar, ya que no se le dan datos etiquetados al algoritmo. El algoritmo simplemente procesa los datos con el objetivo de dar una versión resumida o comprimida de los mismos. Se utilizan generalmente para tareas

de *clustering*: reconocer y agrupar los datos según patrones o similitudes entre ellos.

Existe un tercer grupo de algoritmos con características de aprendizaje de ambos tipos mencionados, denominado **Reinforcement learning**, pero no será utilizado en este trabajo.

2.3.1 El teorema "No free lunch"

¿Existe un algoritmo tal que para un tamaño de $\mathcal{S}$ dado y para todo $\mathcal{D}$ devuelva el h con el menor error? El *teorema "no free lunch"* ("No hay almuerzos gratis" o NFL) para AA (Wolpert 1996) [31] responde esta pregunta: no existe tal algoritmo. Este teorema muestra que todo algoritmo fallará en al menos una tarea en donde otros algoritmos tengan éxito, es decir, ninguno es mejor que otro. Por ello, se debe poseer conocimiento previo sobre el tipo de datos que se tienen y una noción de lo que se quiere obtener para poder elegir $\mathcal{H}$ de manera óptima.

En este trabajo trataremos con tres algoritmos supervisados (*Máquinas de Vectores de Soporte*, *Máquinas de Vectores de Soporte de Una Clase* y *Bosques Aleatorios*) y dos no supervisado (*k-means Clustering* y *Aglomeración de Clusters*). La elección de ellos será justificada más adelante por el tipo de datos que se tienen; en la siguiente sección describiremos el funcionamiento básico de cada uno de ellos.

2.3.2 Algoritmos supervisados

Para realizar la tarea de clasificación se utilizaron algoritmos de aprendizaje supervisado. De forma general, para ejecutar tales algoritmos, es necesario explicitar el conjunto $\mathcal{S}$, especificando cuál es la característica a clasificar y a partir de qué otra(s) debe realizarse tal clasificación. Es decir, debe proveerse al algoritmo con la variable dependiente (datos etiquetados) e independiente(s) (variable(s) predictora(s)). Además, cada algoritmo posee parámetros condicionantes del conjunto $\mathcal{H}$ (y por ende de h), los cuales deben ser elegidos de tal manera que el error $L_S(h)$ sea

mínimo. En esta sección solamente se describirá en líneas generales el funcionamiento de los algoritmos, en el capítulo siguiente se detallarán las variables y los parámetros para ejecutarlos.

Máquinas de Vectores de Soporte

Support Vector Machines (Vapnik 1992) [32] o Máquinas de Vectores de Soporte (llamados *SVM* de ahora en más), son algoritmos que pueden usarse para tareas de clasificación o regresión. Existen diversos *SVM* cuyos métodos de clasificaciones básicas son equivalentes, pero existen variantes de implementación dependiendo de la forma que tenga el conjunto de entrenamiento $\mathcal{S}$. El tipo de $\mathcal{H}$ con el cual trabaja *SVM* se denomina *half-spaces o medio-espacios*, y funciona para tareas de clasificación binaria, como lo es nuestro caso. Está asociado con la definición geométrica de medio-espacio: cualquiera de las dos partes en las que un hiperplano divide al espacio euclídeo.

Sea $\mathcal{X} = \mathbb{R}^d$ e $\mathcal{Y} = \{0, 1\}$. Cada hipótesis de $\mathcal{H}$ se parametriza con un vector $\vec{w} \in \mathbb{R}^d$ y con un $b \in \mathbb{R}$ denominado *término de sesgo*, y al recibir un vector $\vec{x}$ de $\mathcal{X}$ devuelve la etiqueta $(\langle \vec{w}, \vec{x} \rangle + b)$ (ecuación general implícita de un hiperplano). Para el caso $d = 2$ cada hipótesis define una recta o separador que es perpendicular a $\vec{w}$ e interseca el eje vertical en el punto $(0, -\frac{b}{w_2})$. Todo lo que queda del lado de arriba del hiperplano (los puntos que formen un ángulo agudo con $\vec{w}$) son etiquetados como 1 y el resto como 0. Los datos que puedan separarse a través de un hiperplano se denominan linealmente separables.

Existen muchos medio-espacios que pueden dividir los datos, ¿con qué criterio se elige el más adecuado? La distancia entre un punto $\vec{x}$ y el hiperplano definido por $(\vec{w}, b)$ donde $\|\vec{w}\| = 1$ es $|(\langle \vec{w}, \vec{x} \rangle + b)|$. Se define *margen* como la mínima distancia entre un punto del conjunto de entrenamiento y el hiperplano. *SVM* busca separadores de "gran margen": un medio-espacio en donde los puntos del entrenamiento no solo estén bien separados por un hiperplano, sino también lejos de él. Al restringir al algoritmo a

devolver un separador de gran margen se tiene la ventaja de disminuir la complejidad de cálculo cuando se trabaja con espacios grandes.

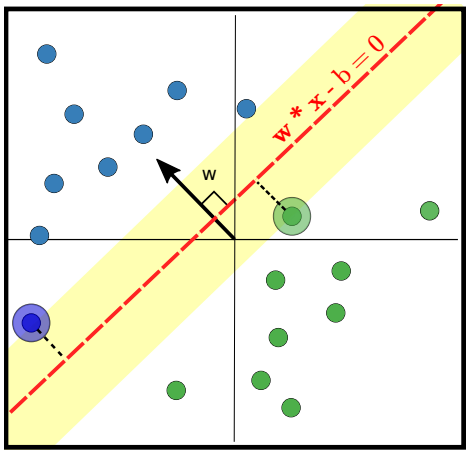

Figura 2.2.: Separación de los datos (clasificados en azul y verde) por medio de un hiperplano (rojo) perpendicular al vector $\vec{w}$ y definido por b, y por los vectores de soporte (azul y verde remarcados con línea de trazos).

El $\vec{w}_0$ que posee el mayor margen tendrá el menor error real. Como el hiperplano de la solución $\vec{w}_0$ de mayor margen se define por medio de distancias, se puede interpretar a los puntos más cercanos al hiperplano como aquellos que lo definen, de ahí el nombre de "vectores de soporte". Los vectores de soporte serán entonces aquellos puntos del conjunto de entrenamiento que estén exactamente a una distancia $\frac{1}{\|\vec{w}_0\|}$ del hiperplano de separación.

Cuando los datos son linealmente separables, se aplica el método sin mayores inconvenientes. Sin embargo, existen datos que no pueden ser separados de manera correcta por medio de un hiperplano, y *SVM* no puede ser utilizado tal como lo describimos. Para resolver esto, se realiza una proyección de los datos a través de una función no lineal a un espacio de mayor dimensión. Los puntos podrán ser separados en este nuevo espacio mediante un hiperplano, y cuando ese hiperplano se vuelva a proyectar al espacio de Entrada, tendrá la forma de una curva no lineal. En principio, siempre es posible encontrar una función que transforme los datos a un espacio de mayor dimensión donde sean linealmente separables. Para optimizar los cálculos de

estas proyecciones, además, se han desarrollado métodos denominados *kernel tricks* o trucos de núcleo. En este trabajo se utiliza el núcleo lineal (el estándar) para tratar casos separables y el núcleo radial para los no separables.

Para información más detallada del funcionamiento de *SVM* se recomiendan los libros de V. Vapnik [33] y B. Schölkopf et al. [34].

Máquinas de Vectores de Soporte de Una Clase

Tradicionalmente, la mayoría de las tareas de clasificación intentan distinguir clases distintas en los datos de evaluación a partir de un entrenamiento etiquetado con estas clases. Existen ciertos problemas, sin embargo, donde sólo hay interés en que se distinga una sola de las clases. Por ejemplo, en tareas como detección de anomalías (mencionada en la Sección 2.1), una propuesta práctica sería entrenar al algoritmo con los datos que pertenezcan a la clase considerada normal o correcta, y luego al testear nuevos datos, el algoritmo distinga los casos que no se asemejen a los del entrenamiento. Schölkopf et al. [35] introdujeron el método de *One Class Support Vector Machines* o Máquinas de Vectores de Soporte de una clase (*OCSVM*) para detectar anomalías con este principio.

El funcionamiento básico es semejante a las transformaciones en el espacio de características de *SVM*: simplemente se busca una función binaria que distinga las regiones en este espacio en donde se encuentra la densidad de probabilidad de la clase deseada. De esta manera, la función devuelve la clase deseada en una pequeña región que contiene los puntos de entrenamiento y etiqueta el resto de los datos como otra clase sin importarle las características que posea.

Se podría decir que el funcionamiento de *OCSVM* es idéntico a *SVM*, puesto que ambos trazan una "frontera" que divide los datos, pero la diferencia radica en que el analista sabe que tiene tipos de datos con ciertas características normales, y quiere identificar los casos que no las poseen. *SVM* no tiene un criterio a-priori que asegure ni que todos los puntos que se saben normales estén en el mismo lado del medio-

espacio, ni que distinga los valores atípicos, puesto que sus márgenes siempre tendrán una tolerancia en la que los (pocos) valores extraños pueden pasar desapercibidos y el error cometido por el algoritmo no cambiará demasiado.

Bosques Aleatorios

Random Forest o Bosques Aleatorios (Breiman 2001) [36] es parte de los denominados *algoritmos de ensamble*: es un ensamble de árboles de decisión.

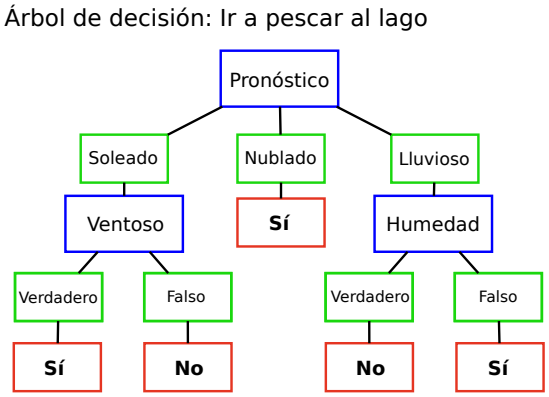

Figura 2.3.: Ejemplo de árbol de decisión: los nodos son las características (cuadros azules) y sus posibles valores en verde, y las hojas las etiquetas con las opciones "Sí", "No" (rojo).

Un árbol de decisión como el de la Fig. 2.3 es un algoritmo $h : \mathcal{X} \to \mathcal{Y}$ que predice la etiqueta asociada con $\vec{x}$ de manera análoga a la estructura de un árbol. El camino de decisión en un árbol comienza en un nodo denominado raíz, a partir del cual realiza una serie de elecciones hasta llegar a una "hoja" del árbol (la etiqueta final). En cada nodo del pasaje raíz-hoja, se exploran las características del objeto $\vec{x}$ a clasificar, y de todas las opciones se elige la que corresponda al objeto. También las opciones pueden estar dadas mediante un conjunto predefinido de reglas de división, como por ejemplo se puede forzar un sesgo hacia un tipo de resultado al otorgar distintos pesos a las opciones. Finalmente, una hoja contiene una etiqueta específica.

El inconveniente con los árboles de decisión es el sobreajuste de los datos de entrenamiento: si se deja que el árbol tenga todas las hojas que se quiera, entonces

es muy posible que cada camino raíz-nodo sea muy específico para un solo objeto, y por lo tanto no sea generalizable para casos ligeramente distintos (es decir, este algoritmo suele tener bajo sesgo pero alta varianza). Existen dos maneras de evitar esto: restringir el tamaño del árbol o hacer un ensamble de árboles. Esto último es, precisamente, el algoritmo *RF*.

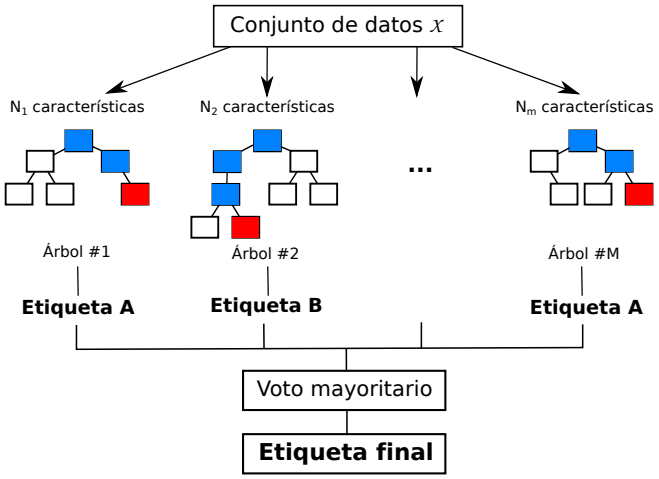

Figura 2.4.: Esquema simplificado de un Bosque Aleatorio. Cada árbol se construye con cada muestreo y N_i características elegidas al azar, lo que permite que se tengan árboles de distintos tamaños. Se define la etiqueta final del objeto por voto mayoritario.

La idea de realizar un "bosque" de árboles permite darle robustez a los resultados de la predicción, es decir, aumentar su exactitud. *RF* es una forma de promediar múltiples árboles de decisión entrenados con distintas partes del conjunto de entrenamiento (Fig. 2.4), con el objetivo de reducir la varianza. Dado un conjunto de entrenamiento $\mathcal{S}$, el algoritmo selecciona M veces (parámetro libre determinado por el conjunto de entrenamiento) una muestra al azar con reemplazo de $\mathcal{S}$ y construye árboles de decisión a partir de cada una de estos muestreos. La predicción de *RF* se obtiene por voto mayoritario sobre todas las predicciones de los árboles individuales. La reducción en la varianza del voto mayoritario queda asegurada, aún si los árboles individuales poseen una varianza alta, siempre y cuando no exista correlación entre ellos. Para asegurar que esto no pase, *RF* introduce en cada división de nodo, un vector de características aleatorio. Si esto no se hiciese, muchos árboles elegirían sólo

aquellas características más significativas a la elección de etiqueta y habría correlación entre ellos. De aquí la aleatoriedad del bosque.

Para más detalles del funcionamiento de *RF* se recomienda la lectura del material de Hastie et al. [37].

2.3.3 Algoritmos no supervisados

Como se mencionó en la Sección 2.3, este tipo de algoritmos se emplean en tareas que no implican generalizaciones, sino simplemente reconocimiento de patrones o similitudes en los datos. Usualmente los resultados que se obtienen de ellos se utilizan para caracterizar datos más que para resolver una tarea del tipo mencionado, ya que no es posible verificar en forma directa si el algoritmo tuvo un buen desempeño.

Para más ejemplos de algoritmos no supervisados se propone la lectura del material de Rokach et al. [38], y para una guía práctica de implementación de *clustering*, se recomienda el libro de Kassambara [39].

K-means clustering

K-means clustering (MacQueen, 1967) [40] es uno de los algoritmos no supervisados de AA de partición más utilizados para agrupar un conjunto de datos con características semejantes. El analista debe especificar el número de grupos k (k *clusters*) en los cuales desea dividir los datos y el algoritmo los agrupará de tal forma que los objetos dentro de un mismo cluster sean lo más similares posible y entre distintos clusters sean lo más disímiles posible. Cada cluster está representado por un centro o centroide, el cual es el promedio de los elementos asignados al cluster.

La idea básica detrás de *k-means clustering* consiste en definir grupos tal que la variación inter-cluster sea mínima. Hay varios algoritmos de *k-means* disponibles, el estándar es el algoritmo Hartigan-Wong (1979), el cual define la variación intercluster como la suma del cuadrado de la distancia (euclídea u otra) entre los ejemplos y su centroide correspondiente.

Se define la variación total inter-cluster (*Total Within Cluster Variation* o $TWCss$) como sigue:

$$TWCss = \sum_{j=1}^{k} W(C_j) = \sum_{j=1}^{k} \sum_{x_i \in C_j} (x_i - \mu_j)^2 \tag{2.1}$$

donde C_j con $j \in \{0, ..., k\}$ es uno de los k clusters, x_i los datos que pertenecen al cluster C_j y μ_j es el promedio de los puntos asignados a este cluster. Esta variación total mide la compactación de los clusters (qué tan bien están divididos), y debe ser la menor posible.

El funcionamiento del algoritmo es sencillo: primero, el analista especifica los k clusters en los que desea dividir sus datos. El algoritmo sitúa en el espacio de datos k puntos al azar, los cuales serán centros provisorios, y calcula la distancia del resto de los datos a cada uno de ellos. Se asigna cada dato al centroide que se encuentre más cercano, y una vez que se tienen los k grupos de datos, se actualizan las posiciones de los centroides al calcular el promedio de los datos del grupo. Al obtener los nuevos promedios, se revisa que cada punto de los datos esté efectivamente más cerca al nuevo centroide de su cluster, caso contrario, se redistribuyen los datos y se calculan los promedios de nuevo.

Este proceso se repite una cantidad de veces especificadas a priori por el analista, o hasta que la variación $TWCss$ de los nuevos centros sea ínfima. De esta manera se obtienen los k clusters deseados.

Las ventajas de *k-means* son su tiempo de cómputo (relativamente corto en comparación con otros algoritmos de partición) y es sencillo de entender. Sin embargo tiene varias desventajas: se necesita conocer a priori la cantidad de clusters k que se desean, y además es muy sensible a datos anómalos y a la elección de los k centros iniciales. Una dificultad relacionada a los datos con valores atípicos es que *k-means* asigna centros que no corresponden a datos que se posean (los centros son promedios) y esto puede conducir a una mala clasificación (Fig. 2.5).

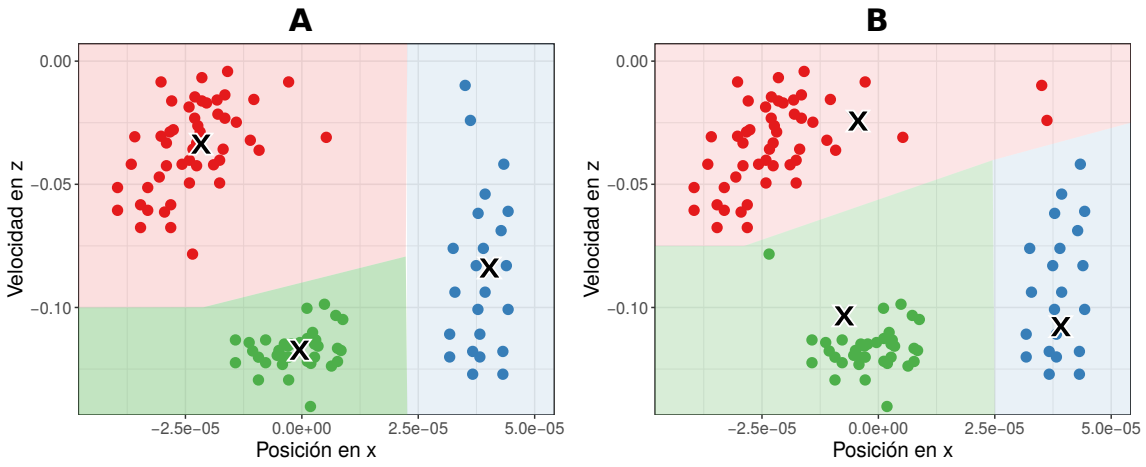

Figura 2.5.: Ejemplo de un buen desempeño de *k-means* (**A**) contra un mal desempeño del algoritmo (**B**), de un espacio de características de partículas clasificadas por su velocidad y posición en el eje z. Las dos clasificaciones (**A** y **B**) son estables, pero la elección al azar de los centroides en **B** perjudica la clasificación.

En el siguiente capítulo se muestra cómo una mala clasificación de *k-means* debido a esta sensibilidad frente a los valores atípicos conduce a la utilización de otro tipo de algoritmo no supervisado: el algoritmo de *Aglomeración de Clusters*.

Aglomeración de Clusters

La Aglomeración de Clusters o *agglomerative clustering* es uno de los tipos más comunes de agrupaciones jerárquicas, utilizado para dividir objetos en clusters basados en sus similitudes. La diferencia con los algoritmos de partición es que no es necesario determinar a priori la cantidad de clusters que se desean obtener.

Este algoritmo inicia tratando cada objeto a clasificar como un cluster individual. Luego, en cada paso, se comparan los clusters de a pares: aquellos que son similares (cercanos) se fusionan en uno sólo, y el proceso se repite hasta que todos los datos estén agrupados en un único cluster. El resultado de las iteraciones posee una estructura denominada dendograma (Fig. 2.6).

Una de las desventajas de la aglomeración de clusters es que el algoritmo no indica en cuántos clusters están divididos los datos, es decir, cuál de las instancias

intermedias de agrupación es mejor. Es por ello que el analista debería estimar en cuantos grupos se dividen los datos, y mediante una función en donde se especifica este número, "cortar" el dendograma de tal forma que queden divididos los datos en los k clusters estimados (Fig. 2.6). El algoritmo devolverá entonces los datos clasificados por pertenencia a cluster.

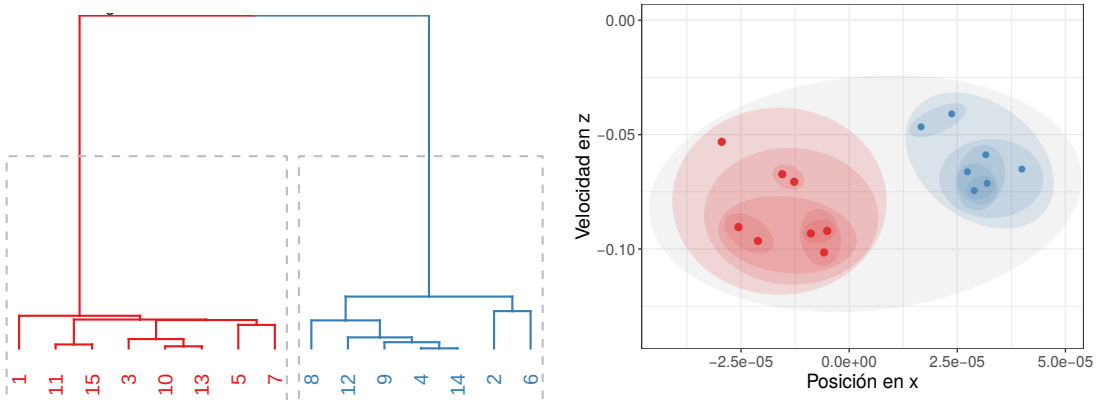

Figura 2.6.: Ejemplo de un dendograma (izq.), donde clasifica 15 objetos en dos clusters de acuerdo a sus distancias relativas en el espacio de velocidades en z y posiciones en z. La figura de la derecha muestra los mismos puntos en el espacio de características, agrupados por jerarquías según sus proximidades.

Aunque es un sistema muy efectivo en ciertas circunstancias (aún donde *k-means* falla), el inconveniente con este algoritmo es que la cantidad de objetos determina la cantidad de cálculos que deben resolverse en el paso inicial. Si se tienen N datos, se tienen N clusters iniciales y se deben calcular $\sum_{n=1}^{N} n - 1$ relaciones sólo en el primer paso. Para volúmenes de datos grandes, el costo computacional puede ser alto.

2.4 La evaluación de h

Una vez que se obtiene de este sistema de AA una función h con el menor error de entrenamiento posible, se debe evaluar la capacidad de generalización efectiva del predictor en datos nuevos, es decir, no pertenecientes al entrenamiento. Este conjunto de datos se denomina *conjunto de evaluación* y debe ser generado por la misma distribución generadora $\mathcal{D}$ del conjunto de dominio $\mathcal{X}$ con el cual se entrenaron los

datos. Para asegurar esto, generalmente los elementos del dominio se dividen en dos partes antes del entrenamiento, una se utiliza para entrenar al algoritmo y la otra para evaluarlo. De esta forma, se asegura que la misma $\mathcal{D}$ genera los dos conjuntos.

Sin embargo, puede suceder que $\mathcal{S}$ y el conjunto de evaluación no sean tomados del mismo dominio (el caso tratado aquí). De ocurrir esto, se puede concluir que un buen desempeño de h como clasificador de tal conjunto de evaluación implica que tanto este como $\mathcal{S}$ efectivamente están generados por la misma distribución $\mathcal{D}$. Por otro lado, si h tiene un mal desempeño, entonces no se puede asegurar si falló alguna parte del algoritmo o si efectivamente los elementos evaluados no son generados por la misma $\mathcal{D}$.

Una vez que se obtienen las predicciones sobre nuevos conjuntos, la manera de evaluar el desempeño de h como clasificador es mediante *métricas de desempeño*. Existen grandes cantidades de métricas y la elección de la más adecuada depende de la tarea que se ha llevado a cabo. En el siguiente capítulo se describirán las métricas específicas utilizadas para este trabajo.

3. MÉTODOS: EL SISTEMA DE AA

En este capítulo se describen las simulaciones de colisiones en las cuales se realizó una clasificación sobre sus partículas, y se caracteriza el sistema de AA utilizado para llevar a cabo tal tarea. Este sistema posee la estructura básica especificada en el capítulo anterior (Cap. 2) en el contexto del problema físico propuesto en la Sección 1.2. Para una referencia representativa de un sistema de AA con algoritmos de Aprendizaje Automático, se recomienda repasar la Fig. 2.1.

3.1 Introducción: Objetivo principal de la tarea de clasificación

La tarea que desea llevarse a cabo en el presente trabajo es una clasificación: etiquetar cada partícula integrante de una simulación de granos porosos colisionantes según pertenencia a cierto fragmento ***luego*** del impacto. Esta identificación de las partículas tiene como objetivo disminuir el tiempo de simulación, en el sentido mostrado en la Fig. 3.1.

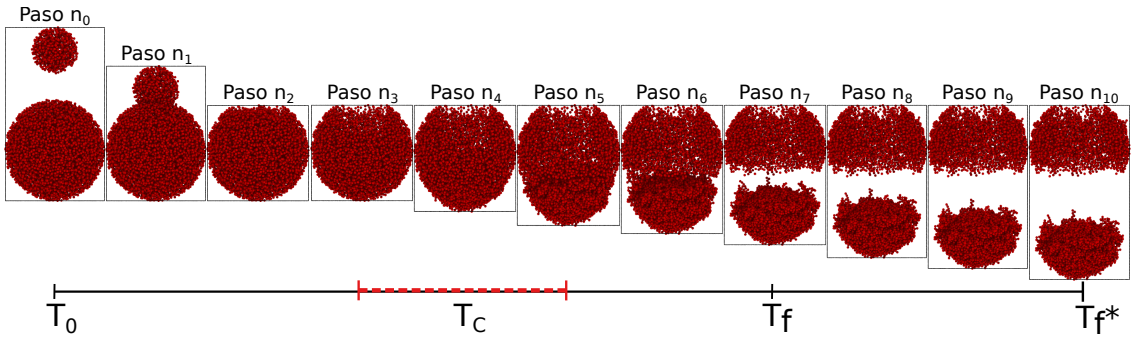

Figura 3.1.: Ejemplo de 11 pasos secuenciales de una simulación de colisión del tipo trabajado, ilustrando el tiempo de finalización total T_f^*, el tiempo en el cual se distinguen visualmente dos fragmentos T_f y el tiempo en el cual se espera que el algoritmo de AA prediga el resultado de la colisión T_c (dentro del intervalo punteado).

Las simulaciones trabajadas finalizan en un tiempo T_f^*, sin embargo a para un tiempo T_f el analista puede determinar visualmente si la colisión es una fragmentación o una aglomeración. Se pretende que el algoritmo supervisado de AA sea capaz de predecir a cual fragmento pertenecerá cada partícula del sistema en un tiempo T_c anterior a T_f, en el cual la distinción visual entre clusters ya no es posible. Para ello se propone armar un sistema de AA como el que se detallará a lo largo de todo este capítulo.

3.2 Simulaciones de colisiones de granos

Todas las simulaciones analizadas en este trabajo poseen características generales similares: dos esferoides, un proyectil y un blanco (inmóvil inicialmente), ambos compuestos por partículas esféricas de SiO_2 idénticas de $R_g = 0.76\,\mu m$ de radio granular, colisionan con un parámetro de impacto de 0 (es decir, son colisiones centrales) a lo largo del eje z. Se les confiere una temperatura a las partículas del blanco, por lo cual adquieren estas una pequeña velocidad total producto de la agitación térmica, y por ende no están totalmente en reposo, como puede observarse en la Fig. 3.2. Sin embargo, esta velocidad es de tres a cuatro órdenes de magnitud menor a la velocidad inicial del proyectil.

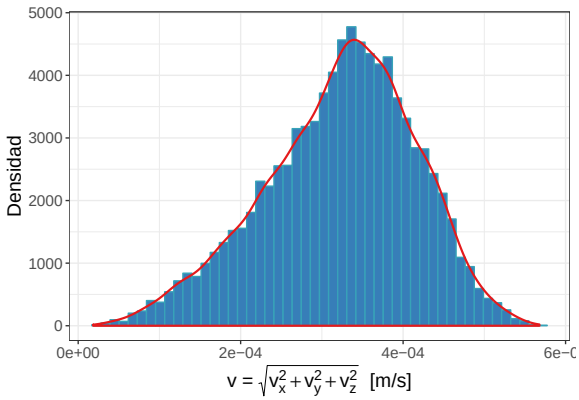

Figura 3.2.: Densidad de las velocidades totales de las partículas constituyentes del blanco, con una media de aproximadamente 3.313e−04$\frac{m}{s}$.

Como resultado de la colisión, el sistema experimentará una aglomeración (proyectil y blanco quedan pegados y sólo se desprende de ellos una fina capa de polvo, compuesta por una pequeña cantidad de dichas partículas) o una fragmentación (el sistema queda dividido en dos o más partes bien diferenciadas), como se muestra en la (Fig. 3.3). Todavía está en discusión cuántas partículas deben quedar en cada fracción para considerar al resultado de una colisión como una fragmentación en vez de una aglomeración. Por ejemplo, se ha catalogado un sistema dividido en alrededor de 33 % de partículas en un cluster y 67 % en el otro como una fragmentación [25]. En este trabajo todas las colisiones se dividen en dos partes o *clusters*, pero no se adentrará en clasificar el tipo de resultado que representan.

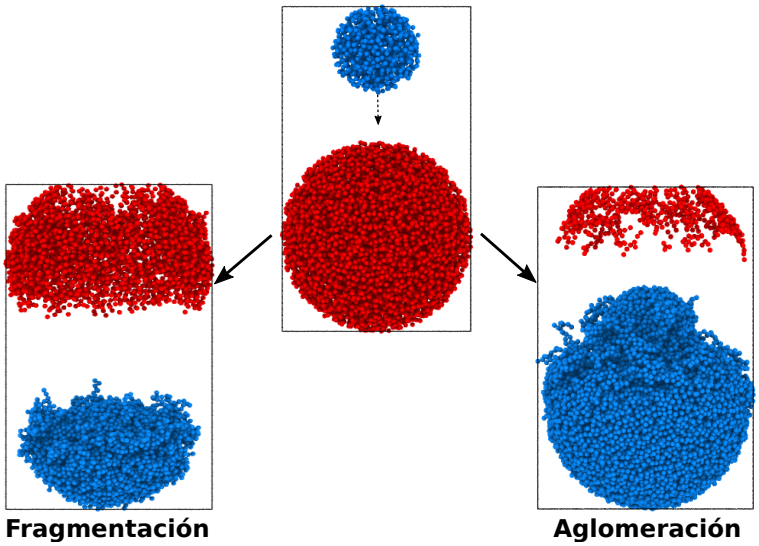

Figura 3.3.: Representación de los dos resultados más frecuentes de una simulación de una colisión.

Existen tres parámetros generales responsables (en parte) de la evolución del sistema: **tamaño de proyectil-blanco**, **factor de llenado** ϕ y **velocidad inicial de proyectil** v_0 (la magnitud de $\vec{v_0}$). El número de partículas en cada simulación depende del tamaño del proyectil-blanco y de ϕ. En la bibliografía [15, 19] se mencionan dos tipos de factores de llenado: el *local* (densidad de partículas dentro de una esfera de radio $N * R_g$, en este caso se utilizó $N = 5$, con centro en la partícula i) y el *global*

(promedio de los factores de llenado locales). En este trabajo ϕ será siempre el factor de llenado local y determina la forma de armado del proyectil y el blanco, y variará durante la colisión debido a la compactación del material durante el impacto [19].

La forma detallada en la cual se arman las muestras para las simulaciones del tipo utilizado fue publicada por Ringl et al. [41].

Utilizando **Dinámica Molecular** (MD) y considerando un modelo de materia granular en el que se incorporan fuerzas de cohesión [15], se realizaron simulaciones como las descritas utilizando el software LAMMPS [1], el cual fue ejecutado principalmente en GPU (*Graphics Processing Unit* o Unidad de Procesamiento Gráfico).

Las simulaciones granulares utilizan datos de entrada generados con un modelo presentado en [41] y un código desarrollado por E. Millán y B. Planes (el primero co-director de este trabajo, ambos integrantes del grupo *SiMAF*). El estilo de par granular utilizado fue desarrollado inicialmente por Ringl et al. [15] para ejecutarse en CPU y posteriormente adaptado por Millán et al. [42] para ejecutarse en GPU NVIDIA con CUDA.

En principio todas las simulaciones se ejecutaron con un *hardware* determinado. Como se mencionó en la introducción (Sección 1.2), no se dispone mejores recursos que los utilizados para ejecutar las simulaciones analizadas en este seminario y por ende no es posible una disminución del tiempo de cómputo mediante mejores equipos. La lista detallada con las especificaciones de hardware y software utilizadas para este trabajo se encuentra en el **Apéndice A**.

En las simulaciones de MD, el tiempo de simulación se discretiza en intervalos denominados pasos temporales. En este caso, cada paso temporal corresponde a $50\,ps$ de tiempo de *simulación*, es decir, el tiempo en el cual transcurre la colisión simulada. El tiempo *real* de simulación es el tiempo reloj que tarda el analista en ejecutar una simulación completa en un equipo y este es el tiempo que se desea disminuir. Al terminar la simulación, el software devuelve un archivo de salida en donde especifica

[1] (`http://lammps.sandia.gov`)

el tiempo real total que le tomó simular el sistema, y por ende, mediante una cuenta sencilla, se puede estimar el tiempo real demorado en ejecutar cada paso.

Se programaron las simulaciones para generar un archivo de salida cada 10000 pasos temporales. Un archivo de salida es un archivo de texto (con un formato similar a CSV o "valores separados por comas") con la configuración del sistema simulado en un determinado paso temporal. Como puede verse en la Fig. 3.4, la descripción de la configuración en cada archivo incluye el **id** (número de identificación de la partícula, fijo para todos los pasos temporales), el **tipo de partícula** (1 ó 2 dependiendo si en el paso temporal 0 la partícula es parte del blanco o el proyectil), las **posiciones en x, y, z**, las **velocidades en x, y, z** y las **velocidades angulares en x, y, z**.

```
ITEM: TIMESTEP
0
ITEM: NUMBER OF ATOMS
11200
ITEM: BOX BOUNDS ss ss ss
4.42885e-07 6.1768e-05
5.47163e-07 6.17254e-05
1.9447e-07 0.000105928
id     type  x           y           z           vx            vy            vz            omegax  omegay  omegaz
1      1     3.105e-05   3.105e-05   3.105e-05   0.000103511   0.000190062   -0.000230885  0       0       0
2      1     2.96799e-05 3.35417e-05 3.04808e-05 -8.32702e-05  -9.33063e-05  -0.000150817  0       0       0
3      1     2.99993e-05 2.925e-05   3.22787e-05 -0.000315167  -0.00032985   -0.000177724  0       0       0
4      1     2.98287e-05 3.02651e-05 2.90546e-05 -9.3444e-05   0.000253729   0.000295921   0       0       0
5      1     2.79869e-05 3.32082e-05 3.03447e-05 -3.97063e-05  -0.000115556  3.65822e-05   0       0       0
6      1     3.0327e-05  3.14704e-05 3.25783e-05 0.000313084   1.92212e-05   -0.000307692  0       0       0
7      1     2.73274e-05 3.09098e-05 2.95345e-05 8.21146e-05   -2.09671e-05  0.000228392   0       0       0
8      1     2.61402e-05 2.85417e-05 2.95511e-05 0.000100833   6.76173e-06   -0.000226314  0       0       0
9      1     2.69086e-05 2.7514e-05  2.69803e-05 -0.000221212  -7.28174e-05  0.000191671   0       0       0
10     1     2.85039e-05 2.90992e-05 2.58412e-05 2.08133e-05   -0.000278903  7.80792e-05   0       0       0
11     1     2.86923e-05 2.84929e-05 3.0529e-05  -0.00031304   -5.15712e-05  -0.000121499  0       0       0
12     1     2.84173e-05 2.98017e-05 2.38092e-05 -0.000155731  0.00015333    0.000164159   0       0       0
13     1     2.9433e-05  2.79447e-05 2.36217e-05 0.000106496   0.000151024   0.000233725   0       0       0
14     1     2.76735e-05 2.80689e-05 2.16433e-05 5.65203e-05   0.000168377   6.9173e-05    0       0       0
15     1     2.64825e-05 2.76125e-05 2.43336e-05 -7.94068e-05  -8.28006e-05  -0.000273674  0       0       0
16     1     2.94416e-05 2.75302e-05 2.53923e-05 0.000247533   -5.45901e-05  5.16764e-06   0       0       0
17     1     2.6891e-05  2.83505e-05 1.91562e-05 -0.00027375   -0.000279206  -0.000328594  0       0       0
.      .     .           .           .           .             .             .             .       .       .
.      .     .           .           .           .             .             .             .       .       .
.      .     .           .           .           .             .             .             .       .       .
11196  2     3.48498e-05 2.11479e-05 8.53518e-05 0             0             -0.1          0       0       0
11197  2     3.29857e-05 2.07483e-05 8.52398e-05 0             0             -0.1          0       0       0
11198  2     3.51746e-05 2.46432e-05 9.98073e-05 0             0             -0.1          0       0       0
11199  2     4.10783e-05 4.0065e-05  9.60989e-05 0             0             -0.1          0       0       0
11200  2     4.15067e-05 3.13053e-05 8.29308e-05 0             0             -0.1          0       0       0
```

Figura 3.4.: Ejemplo de archivo de salida trabajado. En la parte superior se especifica el paso temporal ("TIMESTEP"), el número de partículas ("NUMBER OF ATOMS"), y tamaño de caja ("BOX BOUNDS").

Los tamaños de las simulaciones utilizadas y la memoria que ocupan sus archivos de salida están especificados en el **Apéndice B**.

3.3 La Entrada y su entorno

En esta sección se especifica las simulaciones utilizadas para entrenar a los algoritmos, y a partir de ellas las características generales del conjunto de entrenamiento $\mathcal{S}$ y su entorno (distribución $\mathcal{D}$ generadora de los elementos del dominio $\mathcal{X}$ y la función de etiquetado correcta f).

Simulaciones de entrenamiento

Para los candidatos a conjunto de entrenamiento, se trabajó con una parte de interés del espacio de parámetros generales: un grupo de 15 simulaciones de **tamaño pequeño** (radio de proyectil $\sim 14\,\mu m$ y radio de blanco $\sim 31\,\mu m$, ambos fijos para todas las colisiones), tres **factores de llenado** ϕ (0.15, 0.25, 0.35) y cinco **velocidades iniciales de proyectil** v_0 ($1\frac{m}{s}$, $0.75\frac{m}{s}$, $0.5\frac{m}{s}$, $0,25\frac{m}{s}$, $0.1\frac{m}{s}$) para cada uno de estos ϕ. Cuando se habla de velocidad de proyectil, se refiere a la magnitud del vector de velocidad inicial, el cual tiene un signo negativo porque el proyectil se mueve hacia abajo en el sistema de referencia usado. Estos factores de llenado determinaron el número total de partículas en cada simulación (11200, 18785 y 26206 para $\phi = 0.15, 0.25$ y 0.35 respectivamente), donde alrededor del 9 % del total de partículas pertenecían al proyectil. A estas 15 simulaciones las denominaremos *simulaciones de entrenamiento*, pues de ellas se toman las configuraciones de partículas con las cuales se entrenará los algoritmos. Se eligió este espacio de parámetros como simulaciones de entrenamiento pues ejecutar las 15 simulaciones demora poco tiempo real en comparación a otros tamaños, por tener relativamente menos partículas.

El comportamiento de las simulaciones de entrenamiento es similar: el proyectil se dirige hacia el blanco con una velocidad inicial en z v_0 hasta que impacta con él, y a medida que lo penetra, recibe una fuerza de frenado efectiva resultante de la interacción con las partículas del blanco y entre sus propias partículas [19], lo cual terminará desacelerando a las partículas del proyectil especialmente debido a fuerzas de fricción disipativas. Eventualmente, el sistema se dividirá en dos partes. Una de

ellas será la arrastrada por el proyectil con velocidad en z cuasi-uniforme que tendrá como cota máxima el valor

$$V = \frac{m_p}{m_b + m_p} v_0, \tag{3.1}$$

donde m_p y m_b son las masas del proyectil y del blanco, respectivamente [25]. Notar que para las simulaciones del estilo tratado, $0 < \frac{m_p}{m_b+m_p} < 1$. Para las simulaciones de entrenamiento con el mismo factor de llenado ϕ, la magnitud de V será proporcional a v_0, ya que la relación $\frac{m_p}{m_b+m_p}$ no cambia. Esto implica que las partículas del fragmento arrastrado por el proyectil con velocidad inicial $\vec{v_0} = -1\frac{m}{s}$ tendrán una V mayor a la de los fragmentos finales de simulaciones con $\vec{v_0} > -1\frac{m}{s}$. Para la misma velocidad de proyectil v_0 pero distinto factor de llenado ϕ, se tiene en general que a mayor ϕ menor V (Fig. 3.5).

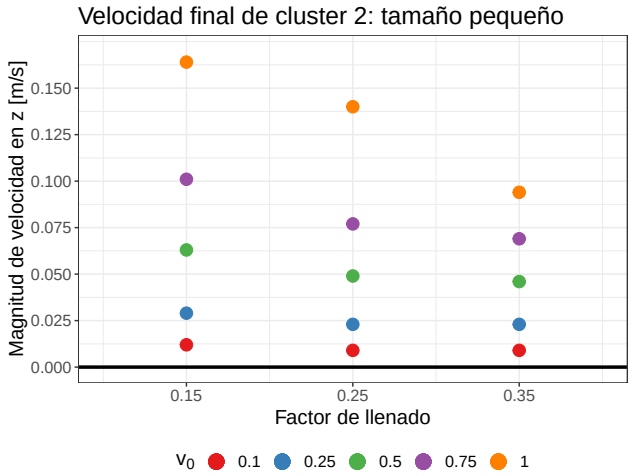

Figura 3.5.: Velocidad final en z del cluster 2 en el paso final de cada simulación dado por el criterio especificado en la sección 3.3.3, para las 15 simulaciones de entrenamiento, para los tres valores de factor de llenado *phi*.

La otra parte en la que queda dividido el sistema es un fragmento originalmente perteneciente al blanco que no es arrastrado por el proyectil, y permanece cuasi-inmóvil (debido a la agitación térmica resultante de la temperatura conferida a las partículas al configurar las simulaciones), como se muestra en la Fig. 3.6. Se verificó para cada simulación trabajada en este seminario que efectivamente el cluster 1 está

compuesto en su totalidad **solo** por partículas del blanco. Se especificará el efecto de este detalle en la parte de resultados. Esta cualidad de las simulaciones es de suma importancia a la hora de elegir el algoritmo para ejecutar la tarea de clasificación, como se verá más adelante. Cabe destacar que si no se da esta particularidad en otro tipo de simulaciones (por ejemplo, otros materiales), esto no afecta en sí al método propuesto, siempre y cuando un cluster permanezca casi en reposo y el otro no.

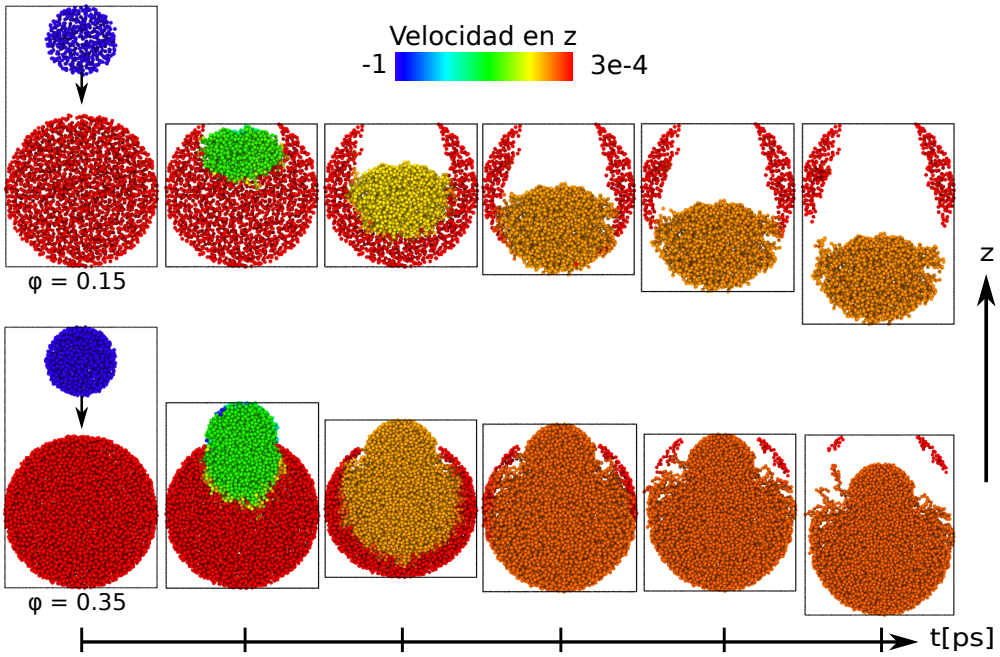

Figura 3.6.: Evolución temporal de una colisión con $v_0 = 1\frac{m}{s}$, $\phi = 0.15$ (arriba) y $\phi = 0.35$ para seis tiempos distintos. Las partículas están coloreadas por velocidad en z. Se ha cortado la muestra en una rodaja de $1e{-}05m$ de espesor (el mismo para los dos ϕ) paralela al eje z, donde se puede apreciar la diferencia de densidad de partículas según ϕ.

Cada una de estas 15 simulaciones de entrenamiento posee una cierta cantidad de archivos de salida que detallan la configuración de cada partícula simulada en un paso temporal dado. Se desconoce a priori qué configuración (de un cierto paso temporal) de cuál simulación (o simulaciones) de entrenamiento es la más indicada para etiquetar y entrenar al algoritmo, tal que éste realice la clasificación de otras configuraciones físicas nuevas (otras simulaciones de distintos parámetros generales).

El criterio para elegir tal configuración puede adquirirse teniendo presente el objetivo de mejorar el tiempo real de simulación. En las siguientes secciones se explorará más a fondo la construcción del conjunto de entrenamiento $\mathcal{S}$.

3.3.1 La distribución $\mathcal{D}$

Tal como se mencionó en la teoría, normalmente se desconoce la función de distribución $\mathcal{D}$, generadora de los elementos del dominio. En el caso de una simulación, se supone en principio que todas las configuraciones especificadas en cada paso temporal por un archivo de salida poseen sus propias distribuciones generadoras. Así, si se tiene una simulación con 1000 archivos de salida, se tendrían en principio $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_{1000}\}$ distribuciones generadoras, una por cada tiempo. Sin embargo, puede suceder que exista alguna distribución $\mathcal{D}_k$ dentro de este conjunto que sea común a varias otras, o que pueda dar cuenta con bastante exactitud de las configuraciones de partículas en otros tiempos. Y no solamente esto, también se podría especular sobre otras simulaciones: ¿Existe alguna $\mathcal{D}_k$ que genere la configuración final de otras simulaciones con distintos parámetros al de la suya propia?

Para verificar la existencia de tal $\mathcal{D}_k$, se supone que existe una simulación de entrenamiento W en la cual se encuentra una configuración de elementos de dominio generada por tal distribución en un paso temporal t_j dado. Se toma tal archivo de salida, se etiquetan las partículas mediante f a fin de obtener el conjunto de entrenamiento $\mathcal{S}$ y se entrena al algoritmo de aprendizaje supervisado con él. Luego, se evalúa la predicción de la función h resultante del algoritmo en configuraciones de diferentes pasos temporales de otras simulaciones distintas de W.

De existir tal conjunto de dominio, capaz de clasificar correctamente partículas de otras simulaciones en distintos tiempos de la colisión según pertenencia a cual fragmento luego del impacto, entonces esta sería una forma de reducción temporal de simulación efectiva: solo se necesita simular W hasta el paso temporal t_f, tomar esta configuración como base para el conjunto de entrenamiento y obtener mediante

el predictor h del algoritmo la clasificación de otras simulaciones según sus partículas pertenezcan a algún fragmento luego de la colisión, sin necesidad de ejecutarlas todas hasta que el sistema esté efectivamente dividido en fragmentos.

3.3.2 El conjunto de dominio $\mathcal{X}$

En vista de hallar la simulación W descrita en la sección anterior entre alguna de las 15 propuestas de entrenamiento, se debe elegir mediante algún criterio la configuración de un paso temporal para ser etiquetado mediante f de forma efectiva. El candidato más seguro es el último archivo de salida que devuelvan las simulaciones de entrenamiento, en el cual se tienen a las partículas bien diferenciadas en dos fragmentos estables. Los elementos del conjunto de dominio $\mathcal{X}$ corresponden entonces a las N partículas constituyentes del proyectil y del blanco de las simulaciones de entrenamiento para el último paso temporal, y pueden representarse como N vectores $\vec{x}_N$ de características (identidad, posiciones, velocidades, etc.)

No todas las características de los elementos influyen en la clasificación. Teniendo presente que simulaciones como las trabajadas poseen la cualidad de que las partículas pertenecientes al blanco, no arrastradas por el proyectil, permanecen cuasi-inmóviles durante todo el proceso, mientras que el proyectil y las partículas arrastradas por él adquieren una velocidad relativamente mayor, se concluye que la velocidad en z de las partículas es un fuerte candidato a ser la variable predictora del algoritmo (Secc. 2.3.2).

Como puede observarse de la Fig.3.7, la variable de velocidad en el eje z es bastante exacta como predictora cuando se la compara con los datos clasificados por pertenencia al cluster en el paso final. Además se destaca nuevamente que la velocidad en z es casi uniforme en todas las partículas arrastradas por el proyectil.

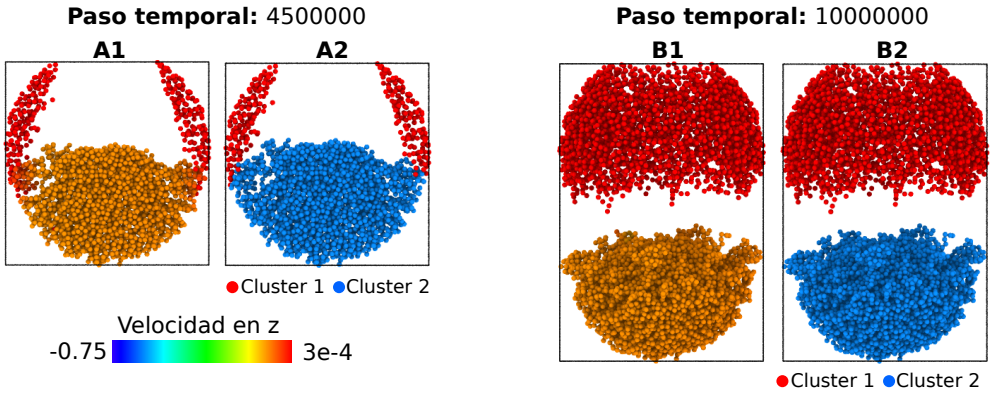

Figura 3.7.: Comparación entre las partículas coloreadas por velocidad (**A1** y **B1**) según la escala especificada en el margen izquierdo, y por pertenencia a cluster en el paso final (**A2** y **B2**), en dos pasos temporales distintos de la misma simulación. Las figuras del paso $4.5e6$ muestran una rodaja del sistema de $1e{-}05m$ de espesor paralela al eje z, para mostrar de forma clara el interior de la colisión.

En conclusión, los elementos del conjunto de dominio a etiquetar serán los de la configuración del paso final de la simulación de entrenamiento escogida. La única característica que poseen las partículas de $\mathcal{X}$ y por medio de la cual el algoritmo clasifica según pertenencia a cluster es la velocidad en z. En el **Apéndice C** se desarrolla en detalle la elección de la velocidad en z como la variable predictora.

3.3.3 La función f de etiquetado: necesidad de algoritmos no supervisados

Para tener el conjunto de entrenamiento completo y "enseñar" a los algoritmos supervisados, es necesario etiquetar a las partículas de $\mathcal{X}$ elegidas según el fragmento al que pertenezcan luego de la colisión. El conjunto de etiquetas asignado por una función f es $\mathcal{Y} = \{1, 2\}$, siendo 1 o 2 el cluster al cual pertenezca la partícula en el paso final de la colisión. Como se mencionó en secciones anteriores, el software de MD también identifica con 1 y 2 las partículas que pertenecen al proyectil y al blanco respectivamente (Fig. 3.4), dichos valores no tienen relación con el etiquetado de las partículas según pertenencia a cluster.

Hasta ahora, el término "paso final de la simulación" se utilizó de manera ambigua. ¿Cuál es el criterio para elegir este paso? Generalmente el analista debe especificar a

priori el paso en el cual la simulación debe terminar, y se asigna un valor de forma estimativa. Usualmente se elige una cantidad de pasos en los cuales se sabe que el sistema no cambiará más su configuración de manera que aporte información adicional. En el tipo de simulaciones con las cuales se trabajaron, se sabe (por experiencia o por inducción) que en un cierto paso el sistema estará muy probablemente fragmentado en dos partes estables, y por si acaso, a veces se programa la simulación para ejecutar un valor mayor a este [25]. Para explorar el alcance de las hipótesis planteadas en el presente trabajo, se necesita uniformizar este tiempo de simulación mediante un criterio de finalización para todas las simulaciones, tanto las de entrenamiento como las que servirán de evaluación, para que el ahorro temporal sea comparable. Esta sección sólo se enfocará en el corte de las simulaciones de entrenamiento.

Se puede realizar el corte de finalización de una simulación de forma manual, por ejemplo utilizando herramientas de visualización como OVITO[2] [43]. El analista determina el momento de corte cuando juzgue visualmente que la separación de los clusters es suficiente como para distinguir los fragmentos. Además de subjetiva, esta es una manera poco práctica. Se busca un criterio de corte que sea automático y universal (particularmente importante cuando se trabajan con varias simulaciones).

Una vez que se posea el paso de finalización de simulación, se necesita etiquetar mediante la función de etiquetado correcto f a las partículas de este paso. Esto también puede hacerse de forma manual (nuevamente contraproducente), pero se busca automatizar también el etiquetado de las partículas. La forma más práctica de hacerlo es utilizando un algoritmo de agrupación, como por ejemplo *k-means clustering*, como función de etiquetado verdadera f.

Para ejecutar *k-means clustering* se necesita también especificar las variables predictoras que mejor separen a las partículas en los grupos que se desean. De la Fig. 3.7, se observa que la velocidad en z es nuevamente una buena elección, pues determina dos fragmentos bien diferenciados en el paso final de la simulación. Para reforzar el etiquetado y generalizarlo aún más, también se utilizó la posición en z del paso

[2] `http://www.ovito.org/`

final como variable predictora, pues las partículas pertenecientes a clusters distintos quedan bien separadas en el espacio.

Utilizando el lenguaje de programación R, se desarrolló un programa para uniformizar el paso de finalización de simulación de las 15 simulaciones de entrenamiento, el cual a su vez también etiqueta los datos de este paso en forma automática. De hecho, el etiquetado es quien determina el tiempo de finalización, como se detalla a continuación.

Teniendo en cuenta que se poseen 15 simulaciones corridas una cantidad subjetiva de pasos, el programa funciona de la siguiente manera: primero, toma el último archivo de salida (el que corresponde al último paso temporal) de cada simulación y utiliza *k-means clustering* para clasificar la pertenencia de cada partícula al cluster 1 o 2 según su posición y velocidad en z. Sea d la distancia vertical entre las partículas del cluster 1 y cluster 2 más cercanas entre sí. Si la distancia d resulta mayor que una distancia de separación R_a previamente escogida de manera heurística (en este caso se eligió $R_a = 6 * 0.76\,\mu m$, 6 veces el radio de una partícula), entonces se toma el paso temporal inmediato anterior y se repite el procedimiento. Si, en cambio, se cumple $d \leq R_a$, entonces la búsqueda se detiene en ese paso y el archivo de salida al cual corresponde pasa a ser el último.

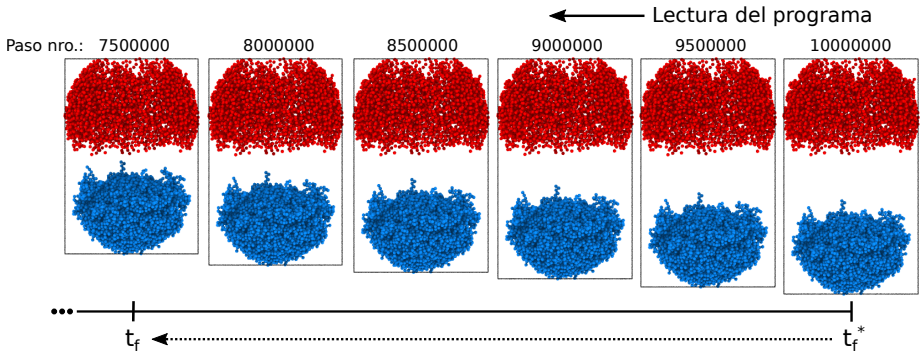

Figura 3.8.: Esquema de lectura del programa armado para clasificar las partículas. El programa comienza a leer a partir del último paso simulado (notar que los fragmentos están excesivamente separados en este paso) y mide la distancia entre ambos clusters. Va descartando los pasos hasta que llega al criterio $d \leq R_a$ y toma este paso como el nuevo t_f.

De esta forma, queda garantizado que todas las simulaciones de entrenamiento terminen en el paso temporal en el cual sus clusters estén separados una distancia de $d \simeq 6 * 0.76\,\mu m$, y que sus partículas estén clasificadas en cluster 1 y 2 según esta configuración final. Esto resulta en 15 conjuntos de entrenamientos $\mathcal{S}_i$, con los cuales se pueden entrenar los algoritmos de aprendizaje supervisado para realizar la clasificación de nuevas simulaciones.

El criterio de corte y el correcto desempeño de *k-means* como clasificador en cada simulación se verifica de forma visual mediante 15 gráficos como el mostrado en la Fig.3.9.

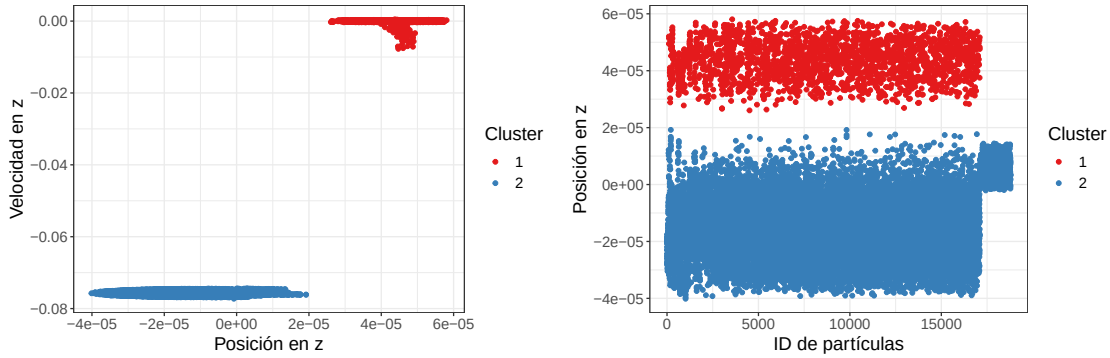

Figura 3.9.: Se muestra un ejemplo de validación visual del paso de finalización de simulación dado por el programa armado, en la simulación de entrenamiento con velocidad de proyectil $\vec{v_0} = -0.75\frac{m}{s}$ y factor de llenado $\phi = 0.25$. En la figura de la izquierda se muestran las partículas en este paso, en el espacio definido por las dos variables predictoras, coloreadas según la etiqueta dada por *k-means*. En la figura de la derecha se muestra la posición en z de las partículas en función de su índice ID, para el mismo paso temporal. Puede verse el proyectil como un bloque (partículas azules aproximadamente formando un cuadrado en el extremo derecho), a cuyas partículas le corresponden valores más altos de ID.

La separación entre los clusters es efectivamente, cercana a $R_a = 6 * 0.76\,\mu m$.

Se tarda aproximadamente 15 días, 23 horas y 19 minutos de tiempo real en simular las 15 simulaciones de entrenamiento (ejecutadas en GPU) hasta la configuración final determinada con este criterio de corte.

Debe tenerse en cuenta que el tipo de programa aquí desarrollado es generalizable solamente a este conjunto de simulaciones de entrenamiento, no se pretende desa-

rrollar una función f utilizando *k-means* que funcione para terminar y etiquetar de forma efectiva las partículas de cualquier simulación. De hecho, la manera en la que actúa el programa está ligado fuertemente a las cualidades de este grupo particular de simulaciones.

3.4 Algoritmos y Salidas

Una vez que se tiene el conjunto de entrenamiento $\mathcal{S}$ (las partículas de $\mathcal{X}$ etiquetadas por f), el siguiente paso es entrenar a los algoritmos de aprendizaje supervisado. En este trabajo se evaluaron tres: *SVM*, *OCSVM*, *RF*. La implementación de estos algoritmos se hace por medio de paquetes especiales de R, detallados en el **Apéndice A**. En esta sección se describe la ejecución de estos algoritmos en R, y en el **Apéndice D** se detallan los hiperparámetros utilizados para cada entrenamiento.

3.4.1 Implementación de *SVM* y *RF*

Se describe a continuación la implementación de los algoritmos *SVM* y *RF* utilizando el paquete **caret** (Clasification And REgression Training)[3] [44] de R.

Validación cruzada de k secciones

Dado un $\mathcal{S}$, cada algoritmo tiene hiperparámetros a ser especificados con el objeto de elegir el predictor h cuyo error de entrenamiento $L_{\mathcal{S}}(h)$ sea el mínimo posible, puesto que este refleja el error verdadero de la predicción. Por ejemplo, *SVM* tiene como hiperparámetros que determinan h el vector $\vec{w}$ y el término de sesgo **b**. Una opción de elección de los hiperparámetros de cada algoritmo es mediante un método denominado *validación cruzada* en donde se toma de los datos a entrenar un *conjunto de validación*; una submuestra sobre la cual se prueban los algoritmos entrenados con

[3] `https://cran.r-project.org/package=caret`

distintas combinaciones de hiperparámetros y luego se elige el predictor que minimice el error sobre este conjunto.

Un tipo de validación cruzada es la validación cruzada de k secciones o *k-fold cross validation* (*cv*). Tomando como ejemplo *SVM*, primero se arma una *grilla de hiperparámetros*, es decir, una combinación de valores de $\vec{w}$ y $\mathbf{b}$, y luego se divide al conjunto de entrenamiento en k partes iguales. Para cada una de estos k subconjuntos de datos, se entrena al algoritmo sobre los subconjuntos restantes usando una de las combinaciones de hiperparámetros tomada de la grilla y luego se calcula el error de entrenamiento sobre la parte que se dejó fuera del entrenamiento (Fig. 3.10). Finalmente, el promedio de los errores obtenidos en cada uno de los k subconjuntos será el error de entrenamiento estimado para ese par de hiperparámetros. Esto se repite con cada una de las combinaciones de la grilla. Una vez que se obtienen los errores de entrenamiento de cada combinación, se elige el par que posee el menor de todos y se entrena todo el conjunto de entrenamiento con ellos.

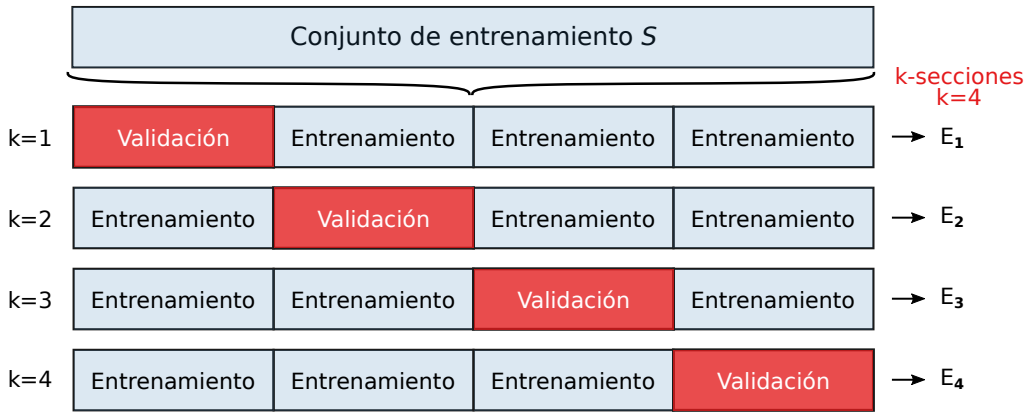

Figura 3.10.: Se muestra un ejemplo de validación cruzada con $k = 4$ secciones para una combinación de hiperparámetros de la grilla. Los errores obtenidos de las 4 iteraciones para esa combinación se promedian, y luego se compara este valor con el promedio resultante de otras iteraciones con otros hiperparámetros para determinar aquellos con el menor error de entrenamiento.

La forma de implementar esta validación sobre los algoritmos en el paquete *caret* es mediante un entrenamiento de control, donde se especifica el método("cv" por *cross*

validation) y el número de secciones k en las cuales se desea dividir en el conjunto de entrenamiento para validar los parámetros, por ejemplo:

```
trainControl(method = "cv", number = 5)
```

De esta forma, se le da un control a los algoritmos con los parámetros que minimizan su error de entrenamiento de forma automática. Cabe destacar que aplicar la validación cruzada en este paquete incluye intrínsecamente sesgar las posibilidades de escoger h a un conjunto $\mathcal{H}$, ya que se le provee al algoritmo una grilla finita con determinados valores de hiperparámetros.

Ejecución de *SVM* y *RF*

El paquete *caret* posee una función de entrenamiento "train", en donde se deben especificar: variables predictoras o independientes (en nuestro caso v_z) y la variable dependiente (cluster al que pertenecen las partículas luego de la colisión, es decir, los elementos del conjunto de entrenamiento), todos los datos de estas variables tomados de los archivos csv como el de la Fig. 3.4, el control de entrenamiento (que contiene la validación cruzada), un preprocesamiento de los datos para estandarizarlos, el método (algoritmo) y la métrica con la cual se desea medir el error en los conjuntos de validación cuando se realice la validación cruzada.

La estandarización de los datos es un ajuste de las variables tal que el promedio de los mismos sea 0 y su varianza sea 1. Se utiliza cuando la distribución de los datos posee forma gaussiana o "campana", para diferenciar aún más valores de distintas escalas en el conjunto de datos. En nuestro caso, la estandarización acentúa la diferencia entre las partículas cuasi-inmóviles pertenecientes al cluster 1 de las que poseen velocidad adquirida por el impacto del proyectil (cluster 2). Para estandarizar los datos, se debe aclarar en la variable *preProcess* los argumentos *center* y *scale*.

Un ejemplo del entrenamiento del algoritmo con todos los elementos nombrados es el siguiente:

```
fit.rad <- train(clust-vz,
                 data = mydata,
                 trControl = ctrl,
                 preProcess = c("center", "scale"),
                 method = "svmLinear",
                 metric = "Accuracy")
```

El método en *SVM* variará según se tengan datos separables o no separables, tal como se describió en la Secc. 2.3.2. Si son separables, el algoritmo será "svmLinear", la forma estándar de *SVM*, y para datos no separables, se utiliza "svmRadial", el cual es *SVM* con el truco de núcleo Radial para transformar los datos. Para usar *RF* simplemente se asigna "rf" en el argumento "method=".

3.4.2 Implementación de *OCSVM*

Se ha afirmado mediante el teorema de "No free lunch" (Secc. 2.3.1) que no existe un algoritmo capaz de desempeñarse de manera óptima en todas las tareas, y la elección del más adecuado requiere un conocimiento de los datos con los que se trabaja. Este es nuestro caso, se sabe que las simulaciones trabajadas poseen un comportamiento característico y es la ya mencionada diferencia en las velocidades relativas de los clusters luego de la colisión. Para cualesquiera parámetros generales de simulaciones elegidos en este tipo de colisiones, en el paso final, la velocidad promedio en z de las partículas del cluster 1 será aproximadamente cero. Por otro lado, las partículas del cluster 2 adquieren en este paso una velocidad en z casi uniforme, distinta a cero (Fig. 3.7). El promedio de la velocidad en z del cluster 2 depende de la velocidad inicial del proyectil v_0: cuanto mayor sea v_0, mayor será su velocidad final promedio en z. Debido a esta dependencia respecto del parámetro general v_0, este promedio de

velocidades cambiará entre simulaciones distintas, pero no así la del cluster 1, puesto que esta siempre será cercana a cero.

Esta cualidad de las simulaciones puede intervenir fuertemente en la elección del algoritmo de aprendizaje: se necesita un algoritmo capaz de aprender que, independientemente de v_0, un cluster siempre terminará con una velocidad promedio en z cercana a cero, y el resto de las partículas que no cumplan esta condición pertenecerán al otro fragmento, sin importar cual sea el valor de velocidad que finalmente posean.

Podría interpretarse esta situación como una tarea de detección de "anomalías" y utilizar el algoritmo *OCSVM* para realizar la distinción entre el conjunto de partículas que se comportan de "forma correcta" (cluster 1), de las que no (cluster 2), ver Sección 2.3.2. Así, se esperaría que este algoritmo sea capaz de distinguir siempre partículas pertenecientes al cluster 1 de forma correcta, clasificando al resto como pertenecientes al cluster 2, independientemente de los parámetros generales de la simulación que se esté evaluando. Notar que *OCSVM* no funcionaría si, por ejemplo, el sistema se fragmentara en agrupaciones de partículas cuya configuración final varíe según los parámetros generales de la simulación, que es lo que ocurre a velocidades mucho mas altas [19].

Para implementar el algoritmo *OCSVM* se necesita otro paquete de R, denominado **e1071**[4] [45], el cual posee una función de entrenamiento del algoritmo *svm()* en la cual debe especificarse que se realiza la clasificación mediante "una clase" (*OCSVM*). A esta función se le dan los parámetros previamente obtenidos de una grilla de validación también incluida en el paquete. A diferencia de *SVM* y *RF*, se debe entrenar al algoritmo sólo con la clase de interés (partículas que pertenezcan al cluster 1), aunque se especifican las mismas variables predictoras y dependiente que estos.

[4] `http://cran.rproject.org/web/packages/e1071/index.html`

3.4.3 Conjunto de evaluación: Extensión local y distante

Al entrenar a los algoritmos supervisados nombrados con alguna combinación elegida de las simulaciones de entrenamiento, se obtiene finalmente una función predictora h. Repasando los objetivos de la tarea de clasificación especificadas en la Sección 3.3.1, la función h debe clasificar partículas no solamente de configuraciones finales de otras simulaciones, sino también aquellas de tiempos anteriores.

Se probó la capacidad de h de generalizar en dos conjuntos de simulaciones, diferenciadas por el parámetro general de tamaño de proyectil-blanco. Se elige este parámetro como divisor de las simulaciones de evaluación porque a mayor tamaño, mayor complejidad de cálculo computacional (pues hay más partículas a simular) y por ende, mayor tiempo real de simulación. Uno de los objetivos principales de la tarea es disminuir este tiempo.

El primer conjunto de simulaciones evaluadas por h corresponde al tamaño pequeño (radio de proyectil $\sim 14\mu m$ y radio de blanco $\sim 31\mu m$), y son las mismas 15 simulaciones de las que se pueden elegir los conjuntos de entrenamientos. La evaluación de este grupo se denomina *extensión local*, e implica observar qué tan bien clasifica h en simulaciones del mismo tamaño que las entrenadas, pero con v_0 y ϕ distintos al del conjunto de entrenamiento.

El otro grupo corresponde a dos tamaños de proyectil-blanco más grandes que el pequeño, denominados "mediano" y "grande". Del tamaño mediano se poseen 15 simulaciones de los mismos v_0 y ϕ que las pequeñas: tres factores de llenado ϕ (0.15, 0.25, 0.35) y cinco velocidades iniciales de proyectil ($1\frac{m}{s}$, $0.75\frac{m}{s}$, $0.5\frac{m}{s}$, $0,25\frac{m}{s}$, $0.1\frac{m}{s}$) combinadas. También se mantiene el mismo radio del proyectil, pero el radio del blanco es de $\sim 44\mu m$. El proyectil corresponde entonces al 3 % del total de las partículas. Del tamaño grande sólo se tienen cinco simulaciones correspondientes a velocidades v_0 de proyectil $1\frac{m}{s}$, $0.75\frac{m}{s}$, $0.5\frac{m}{s}$, $0,25\frac{m}{s}$ y $0.1\frac{m}{s}$, pero solamente $\phi = 0.15$. El radio del blanco es de $\sim 52\mu m$. Al mantener el mismo radio de proyectil que los otros tamaños, las partículas del proyectil representan el 2 % del total de partículas. El

objetivo de evaluar h sobre estos dos tamaños es clasificar las partículas en simulaciones que demoran más tiempo por ser más complejas. La evaluación de este grupo de simulaciones de dos tamaños se denomina *extensión distante*.

3.4.4 Etiquetado de los conjuntos de evaluación

Etiquetar los datos es importante no sólo para la etapa de entrenamiento, sino también para la de evaluación de desempeño: una vez que se obtiene la predicción de h sobre el conjunto de evaluación, se deben comparar los valores predichos con las verdaderas etiquetas de pertenencia final de las partículas a un cluster. De esta comparación se obtiene qué tan bien clasificó el algoritmo en cluster 1 y cluster 2 a los nuevos datos.

Para las simulaciones de tamaño pequeño, se usaron las etiquetas obtenidas por medio de *k-means* (Secc. 3.3.3). Sin embargo, las simulaciones de la extensión distante poseen partículas con valores atípicos en varios pares v_0, ϕ que interfieren con la clasificación de *k-means* y se obtienen resultados como el de la Fig. 3.11. Por lo tanto, para poder etiquetar bien las simulaciones del tamaño más grande, se utilizó, en vez de *k-means*, el algoritmo de Aglomeración de Clusters en el programa descrito en la Sección 3.3.3. Se mencionó anteriormente que este algoritmo caracteriza mejor los valores atípicos, pero computacionalmente es bastante costoso cuando analiza una cantidad de datos como los planteados en las simulaciones de tamaño mediano y grande (entre 31087 y 120894 partículas).

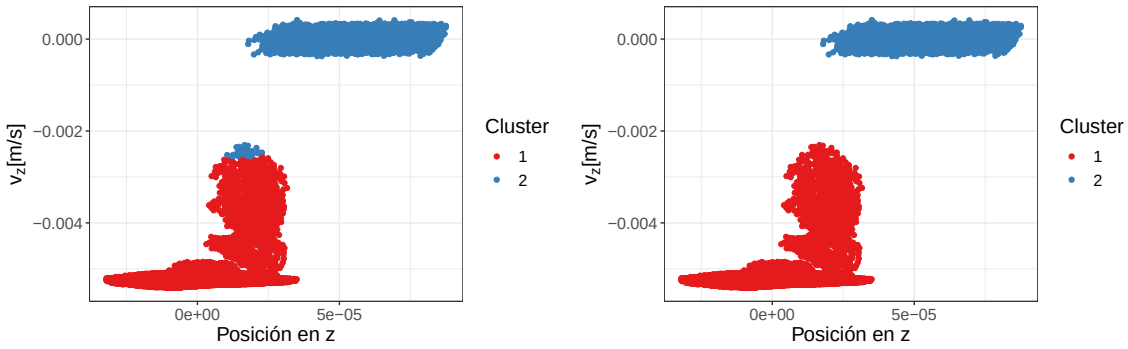

Figura 3.11.: Se muestra una imagen de una mala clasificación de *k-means* (izq.) en la simulación de tamaño mediano con $v_0 = 0.1\frac{m}{s}$ y $\phi = 0.15$, y la clasificación del algoritmo de Aglomeración de Clusters para el mismo paso temporal de este mismo caso (der.).

R contiene dos paquetes que optimizan el tiempo de cómputo del Algoritmo de Aglomeración: *parallelDist* y *fastCluster*. El paquete *parallelDist* permite calcular en paralelo la matriz de distancias entre datos, utilizando todos los núcleos CPUs disponibles, agilizando el tiempo de cálculo y optimizando el uso de la memoria de la máquina en donde se realicen los cálculos. Por otra parte, el paquete *fastCluster* permite ejecutar el algoritmo de forma eficiente para tamaños de datos muy grandes. Como referencia, para analizar un archivo de salida de una simulación de tamaño mediano con $\phi = 0.15$ ($\sim$31000 partículas), las funciones estándar *dist* (para calcular la matriz de distancias entre datos) y *hclust* (para ejecutar el algoritmo de Aglomeración), utilizando un núcleo CPU FX-8350, demoran 78s, mientras que *fastCluster* y *parallelDist* demoran 27s (en 8 núcleos CPUs FX-8350). Esto es alrededor de un 2.88x veces mas rápido que las funciones estándar. Para una simulación de tamaño mediano con $\phi = 0.35$ ($\sim$72353 granos), *hclust* no puede ejecutarse debido a que posee un límite de $\sim$65536 granos que puede calcular, sin embargo *fastCluster* y *parallelDist* son ejecutables para este tamaño y demoran 106s en procesar un archivo de salida.

Mediante este algoritmo se logró clasificar los datos de las simulaciones más grandes para poder evaluar el desempeño de las predicciones, además de obtener el tiempo final de simulación para el paso en el cual la separación de los clusters es de $d \simeq 6 * 0.76\,\mu m$.

3.4.5 Matriz de Confusión y mediciones de desempeño

Para poder evaluar la capacidad de una función h de generalizar, debemos medir cuantitativamente su desempeño. Usualmente la elección de la mejor medición de desempeño es específica para la tarea llevada a cabo por el sistema y aunque puede resultar directa y objetiva, casi nunca es trivial.

Para una tarea de clasificación binaria como la presente, una manera de definir y visualizar el desempeño de los algoritmos es mediante una *Matriz de Confusión* (MC), una tabla que categoriza predicciones de acuerdo a su correspondencia con el etiquetado verdadero de los datos. Una de las dimensiones de la tabla indica la cantidad de etiquetas predichas, es decir, cuántos elementos corresponden a una categoría u otra según el algoritmo, mientras que la otra dimensión indica cuántos elementos verdaderamente corresponden a esas categorías.

Generalmente se distingue una clase de otra mediante los términos "positiva" y "negativa", aunque en el caso presentado es indistinto cuál es la etiqueta positiva o negativa. Elegimos el cluster 1 (fragmento superior) como la clase positiva. La relación entre las predicciones de clases positiva y negativa (cluster 2) puede ser visualizada en una Matriz de Confusión 2 (Fig. 3.12) que muestra si las predicciones pertenecen a alguna de estas categorías:

1 **Verdaderos Positivos (VP)**: Datos correctamente clasificados como perteneciente al cluster 1.
2 **Verdaderos Negativos (VN)**: Datos correctamente clasificados como perteneciente al cluster 2.
3 **Falso Positivo (FP)**: Datos incorrectamente clasificados como perteneciente al cluster 1.
4 **Falso Negativo (FN)**: Datos incorrectamente clasificados como perteneciente al cluster 2.

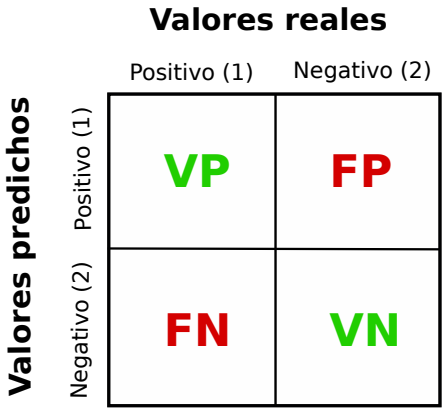

Figura 3.12.: Matriz de Confusión: en la diagonal se encuentran todos los valores bien clasificados.

A partir de estas cuatro categorías, se definen las siguientes métricas de desempeño:

- **Exactitud:** también llamada tasa de éxito, se define formalmente como

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN}, \tag{3.2}$$

  donde los términos VP, VN, FP y FN se refieren al número de veces en el que la predicción cae en alguna de estas categorías. La exactitud es entonces la proporción que representa el número de verdaderos positivos y negativos sobre el número total de predicciones.

  La **tasa de error** es la proporción de ejemplos mal clasificados, dada por

$$tasa\ de\ error = \frac{FP + FN}{VP + VN + FP + FN} = 1 - Exactitud. \tag{3.3}$$

- **Sensibilidad:** en nuestro caso, mide la proporción de ejemplos pertenecientes al cluster 1 correctamente clasificados:

$$sensibilidad = \frac{VP}{VP + FN}. \tag{3.4}$$

- **Precisión:** es la proporción de ejemplos pertenecientes al cluster 1 que verdaderamente pertenecen a ese cluster; un modelo preciso sólo predecirá la pertenencia al cluster 1 en casos en los que sea muy posible que lo sea. Da una noción de confianza en los resultados.

$$precisión = \frac{VP}{VP + FP}. \tag{3.5}$$

Las métricas aquí descritas pueden tomar valores entre 0 y 1, siendo 0 el valor más bajo y 1 el más alto.

3.5 Porcentaje de ganancia temporal

Uno de los resultados principales que se desea obtener es la disminución de tiempo de simulación al encontrar alguna configuración de partículas que sea capaz, luego de entrenar a un algoritmo, de caracterizar la configuración final de otras simulaciones. Para cuantificar la disminución temporal que conlleva la elección de un cierto conjunto de entrenamiento, se deben tener en cuenta dos tiempos clave: el primero es el tiempo real de finalización de simulación T_f (obtenido del último paso temporal determinado mediante el criterio descrito en la Seccs. 3.3.3 y 3.4.4). El segundo es el tiempo para el cual el algoritmo A logra distinguir los dos fragmentos finales, con una cierta exactitud. A este tiempo lo denominaremos tiempo de corte optimizado T_c. Por supuesto se quiere que T_c sea el menor posible, para tener una mayor ganancia temporal.

La elección de T_c es, en parte, relativa al analista. Este tiempo no necesariamente debe tener la clasificación de las partículas de forma 100 % exacta: a veces, al analista le basta distinguir los clusters con un 90 % de exactitud para determinar el tipo de colisión que se produjo, y ese T_c sería menor que aquel con, por ejemplo, un 99 % de exactitud.

El porcentaje de ganancia ahorrado se calcula de la siguiente manera: sea $\mathcal{S}_s$ el conjunto de entrenamiento que se utiliza para entrenar el algoritmo A. Asumimos

que se tienen s_n simulaciones distintas, es decir, distintas combinaciones de tamaños proyectil-blanco, velocidades iniciales de proyectil v_0 y factor de llenado ϕ. El conjunto $\mathcal{S}_s$ está formado por las configuraciones finales de $s = \{s_1, s_2, ..., s_m\}$, con $m < n$ (en nuestro caso, todas de tamaño pequeño). Cada $s_j \in s$ debe ejecutarse hasta T_{f_j} (tiempo real de finalización de simulación para s_j), puesto que se necesita la configuración del último paso etiquetado para poder entrenar, por lo tanto el tiempo total consumido para obtener $\mathcal{S}_s$ está dado por

$$T_e = \sum_{j=1}^{m} T_{f_j} \tag{3.6}$$

donde cada T_{f_j} representa el tiempo final de la simulación s_j utilizada en el entrenamiento. A esta sumatoria de tiempos la denominaremos *tiempo de entrenamiento* T_e, e indica el tiempo real demorado en ejecutar todas las simulaciones utilizadas en el entrenamiento, cada una hasta su T_{f_j}.

Si $\mathcal{S}_s$ da cuenta del resto de las simulaciones s_k con $k \in \{m+1, ..., n\}$, es decir, si logra predecir el desenlace de estas colisiones a un tiempo $T_{c_k} < T_{f_k}$ entonces el tiempo total consumido para ejecutar estas s_k será

$$\sum_{k=m+1}^{n} T_{c_k} \tag{3.7}$$

De esta manera, el tiempo total real de simulación requerido para obtener la configuración final de todas las simulaciones $s_1, ..., s_n$ es

$$T = T_e + \sum_{k=1+m}^{n} T_{c_k} = \sum_{j=1}^{m} T_{f_j} + \sum_{k=1+m}^{n} T_{c_k} \tag{3.8}$$

Finalmente, si T_n es el tiempo total real que demoraría ejecutar las n simulaciones hasta sus respectivos T_f, es decir:

$$T_n = \sum_{l=1}^{n} T_{f_l}, \tag{3.9}$$

entonces se define el *porcentaje de ganancia temporal* como

$$G_t = \left(1 - \frac{T}{T_n}\right) * 100. \tag{3.10}$$

T_n variará en la extensión local y distante, pues se prueban distintos tamaños de simulaciones que conllevan tiempos de finalización distintos.

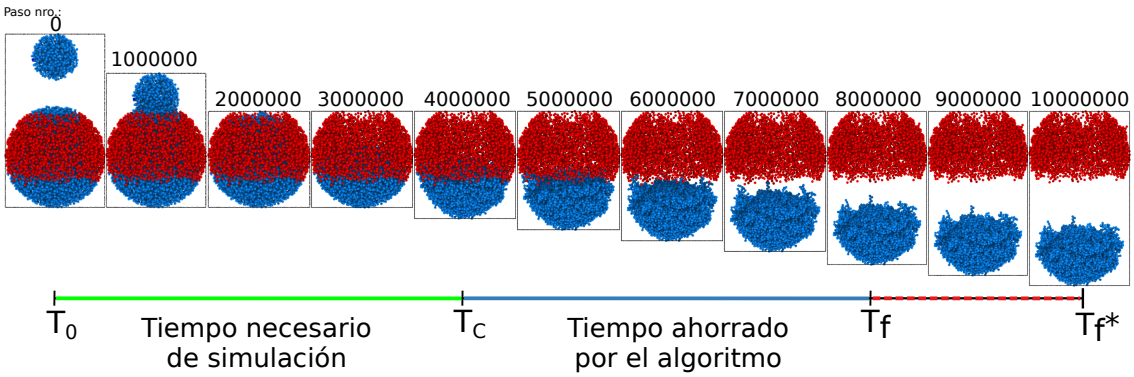

Figura 3.13.: Se muestran 11 pasos temporales de la simulación de $v_0 = 1\frac{m}{s}$ y $\phi = 0.15$. Se esquematizan el tiempo de finalización dado por el analista (T_f^*), el cual se desecha por el tiempo de finalización T_f dado por el algoritmo no supervisado. El tiempo de corte T_c dado por el algoritmo supervisado es aquel en el cual se puede saber el desenlace de la colisión con una cierta exactitud. Como mínimo el analista necesitará ejecutar la simulación hasta este tiempo. Las partículas están coloreadas por pertenencia final a cluster.

En la Fig. 3.13 se muestran los tiempos descritos en esta Sección. El objetivo de entrenar a los algoritmos mediante un cierto conjunto $\mathcal{S}$ es obtener una predicción del desenlace de la colisión a un tiempo T_c que permita ahorrar el tiempo en el intervalo $T_f - T_c$.

Los distintos entrenamientos y las pruebas realizadas para tal fin se desarrollan en el siguiente capítulo.

4. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan los resultados de distintas pruebas de clasificación a través de distintos conjuntos de entrenamiento y algunas conclusiones principales. Cada prueba realizada tiene doble objetivo: por un lado, encontrar cuales conjuntos de entrenamiento $\mathcal{S}$ tomados de las simulaciones de entrenamiento entrenan a los algoritmos para dar predictores generalizables a todas las simulaciones (de tamaño pequeño primero, y luego mediano y grande), y por el otro, para aquellos conjuntos efectivamente generalizables, calcular la ganancia temporal en tiempo de simulación de las predicciones de los algoritmos, mediante la fórmula 3.10.

Se presenta también el tiempo real empleado en entrenar a los algoritmos mediante estos $\mathcal{S}$, leer los archivos de salida, y predecir y evaluar los entrenamientos utilizados. También se realiza un análisis del desempeño del algoritmo *OCSVM* y se dedica una sección a la estimación de tiempo de finalización de simulaciones nuevas mediante extrapolación de los datos.

4.1 Introducción: consideraciones generales

Para entrenar a los algoritmos se utilizaron un total de once combinaciones de las configuraciones finales de las 15 simulaciones de entrenamiento (Secc. 3.3). Se armaron entrenamientos por ϕ, v_0 y una combinación de ambas, tal como se muestra en la Fig.4.1.

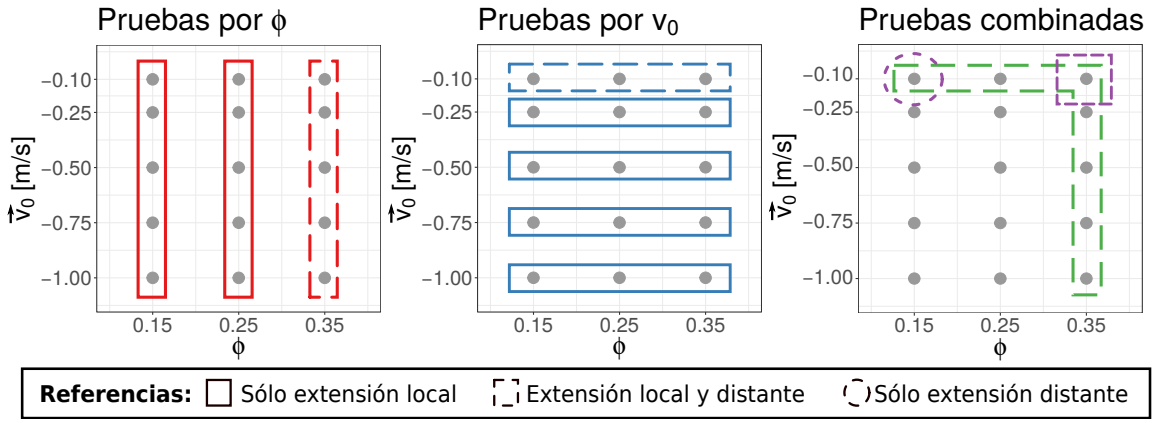

Figura 4.1.: Se muestran en recuadros y círculos los once entrenamientos utilizados en todo el capítulo. Cada punto gris simboliza la configuración final de una simulación de tamaño pequeño con parámetros generales (v_0, ϕ). Los entrenamientos se armaron como combinaciones de estas configuraciones. En las referencias se indican si el entrenamiento fue evaluado sobre simulaciones de tamaño pequeño (extensión local), sobre simulaciones de tamaño mediano y grande (extensión distante), o ambas.

Por comodidad, a cada simulación caracterizada por el par (v_0, ϕ) la denominaremos *caso*. Los entrenamientos especificados en la Fig.4.1 fueron evaluados en los siguientes casos:

- **Extensión local:**
  - Simulaciones de tamaño pequeño $(0.1\frac{m}{s}, 0.15)$, $(0.25\frac{m}{s}, 0.15)$, $(0.5\frac{m}{s}, 0.15)$, $(0.75\frac{m}{s}, 0.15)$, $(1\frac{m}{s}, 0.15)$, $(0.1\frac{m}{s}, 0.25)$, $(0.25\frac{m}{s}, 0.25)$, $(0.5\frac{m}{s}, 0.25)$, $(0.75\frac{m}{s}, 0.25)$, $(1\frac{m}{s}, 0.25)$, $(0.1\frac{m}{s}, 0.35)$, $(0.25\frac{m}{s}, 0.35)$, $(0.5\frac{m}{s}, 0.35)$, $(0.75\frac{m}{s}, 0.35)$, $(1\frac{m}{s}, 0.35)$.
- **Extensión distante:**
  - Simulaciones de tamaño mediano $(0.1\frac{m}{s}, 0.15)$, $(0.25\frac{m}{s}, 0.15)$, $(0.5\frac{m}{s}, 0.15)$, $(0.75\frac{m}{s}, 0.15)$, $(1\frac{m}{s}, 0.15)$, $(0.1\frac{m}{s}, 0.25)$, $(0.25\frac{m}{s}, 0.25)$, $(0.5\frac{m}{s}, 0.25)$, $(0.75\frac{m}{s}, 0.25)$, $(1\frac{m}{s}, 0.25)$, $(0.1\frac{m}{s}, 0.35)$, $(0.25\frac{m}{s}, 0.35)$, $(0.5\frac{m}{s}, 0.35)$, $(0.75\frac{m}{s}, 0.35)$, $(1\frac{m}{s}, 0.35)$.
  - Simulaciones de tamaño grande $(0.1\frac{m}{s}, 0.15)$, $(0.25\frac{m}{s}, 0.15)$, $(0.5\frac{m}{s}, 0.15)$, $(0.75\frac{m}{s}, 0.15)$, $(1\frac{m}{s}, 0.15)$.

Para evaluar cada uno de estos casos, los algoritmos entrenados clasificaron las configuraciones de 100 pasos temporales uniformemente espaciados de cada uno, y el desempeño de la tarea fue determinado mediante las tres métricas mencionadas en la Sección 3.4.5: exactitud, precisión y sensibilidad. Diremos que el desempeño de un algoritmo es ***correcto*** cuando la exactitud medida en cada paso describe una curva suave y creciente aproximadamente sigmoidea. Se espera un comportamiento de esta forma por la ecuación de la exactitud (Ec. 3.2): antes de la colisión la exactitud siempre es la menor posible debido a la disposición del blanco y el proyectil, bastante distinta a la forma final del sistema, como detallaremos más adelante. Luego de la colisión, a medida que se separan los fragmentos, los algoritmos empiezan a distinguir las partículas por su velocidad en z v_z y la curva de exactitud crece rápidamente hasta que se estabiliza en los valores máximos, una vez que los dos clusters se separan. Si la curva experimenta un corte abrupto en su evolución o un comportamiento irregular es muy probable que se deba a un sobreajuste del predictor.

Se espera encontrar algún (o algunos) entrenamientos cuyo desempeño sea correcto en todos los casos evaluados dentro de cada extensión, es decir, que sea capaz de clasificar "correctamente" (sujeto a un pequeño error) la configuración final de cada caso antes de finalizar la simulación. De esta manera, se logra generalizar el aprendizaje y consecuentemente se logra ahorrar tiempo de simulación.

La organización de las secciones que siguen se detalla a continuación. Primero, se realizaron pruebas preliminares y generales para controlar el comportamiento de los algoritmos, para corroborar que clasifican conforme se espera de ellos. Seguidamente, se exploró el espacio de parámetros de las simulaciones de tamaño pequeño con los entrenamientos detallados en la Fig. 4.1 (extensión local) y luego se extendieron algunos de estos entrenamientos a simulaciones de tamaños mayores (extensión distante). Se presenta también el tiempo real que demora la implementación del sistema de AA armado: el entrenamiento de los algoritmos y la lectura de cada archivo de salida de los casos evaluados, la cual incluye las predicciones y evaluaciones de los entrenamientos utilizados. También se realiza un análisis del desempeño del algoritmo *OCSVM* y se

dedica una sección a la estimación de tiempo de finalización de simulación mediante extrapolación de los datos.

4.2 Pruebas preliminares

Se realizó una primera prueba de control con las simulaciones de entrenamiento, que consistió en entrenar los tres algoritmos en cada caso de tamaño pequeño y evaluarlo consigo mismo. En esta prueba, la evolución de los tres algoritmos es correcta, como muestra la Fig.4.2. El eje en x está ajustado mediante una escala logarítmica, puesto que, como la cantidad de pasos de simulación varía dependiendo de los valores de los parámetros v_0 y ϕ, una escala lineal no permite distinguir correctamente la forma de la evolución temporal, en particular en el caso de las velocidades mayores.

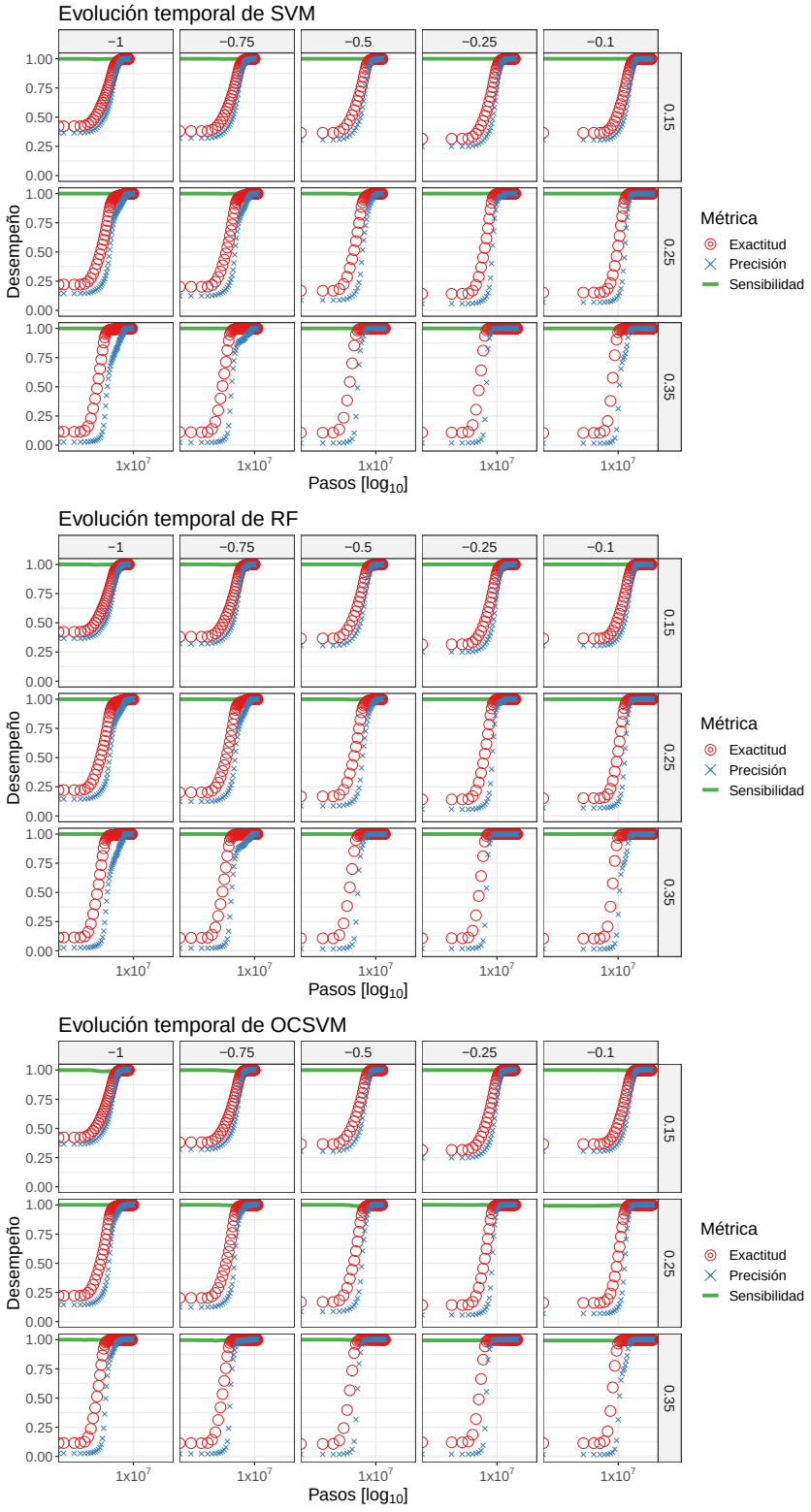

Figura 4.2.: Desempeño de las métricas exactitud, sensibilidad y precisión en función del número de pasos temporales para cada caso de tamaño pequeño entrenado y testeado consigo mismo. Las columnas representan las $\vec{v_0}$ y las filas, los ϕ.

Como puede observarse, los tres algoritmos poseen un comportamiento en función del paso temporal semejante para las tres métricas. En los primeros pasos temporales, los algoritmos tienen poca precisión (valores clasificados como cluster 1 que verdaderamente pertenecerán al cluster 1), mientras que la sensibilidad (partículas clasificadas correctamente como cluster 1) permanece casi uniforme en todos los pasos. Este fenómeno puede entenderse observando la proporción de partículas que el algoritmo clasifica como VP (verdaderos positivos), VN (verdaderos negativos), FP (falsos positivos) y FN (falsos negativos), definidos en la Sección 3.4.5), como muestra la Fig.4.3.

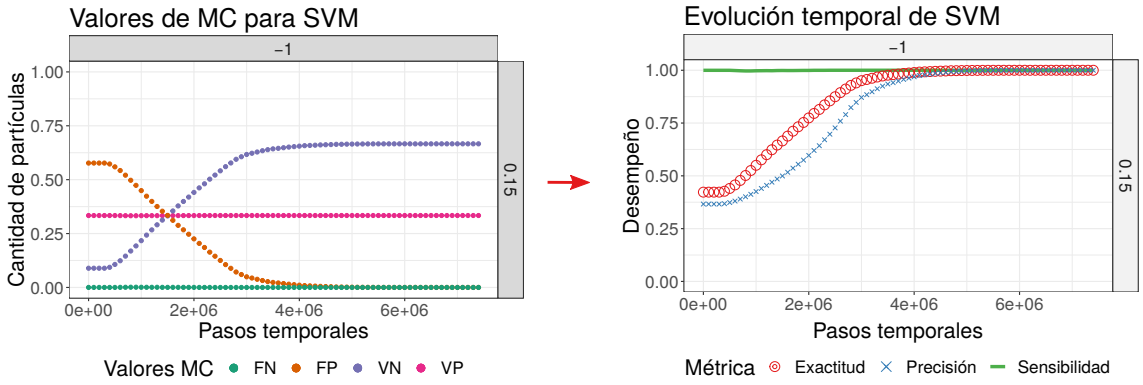

Figura 4.3.: Se muestran los valores de la MC (Matriz de Confusión) VP, VN, FP y FN para cada paso temporal de la implementación de *SVM* (*RF* y *OCSVM* se comportan de manera similar) en el caso $(1\frac{m}{s}, 0.15)$. De la combinación de estas cuatro cantidades se calculan las métricas mostradas en los gráficos de la derecha. Los valores de la MC se encuentran normalizados. Notar que los pasos temporales están en escala lineal.

De esta figura se observa que los FP y FN tienen un comportamiento inverso a medida que aumentan los pasos temporales. En el paso cero, para *SVM* (y los otros dos) más de 0.5 de las partículas son clasificadas como que pertenecerán al cluster 1, cuando en realidad pertenecerán al cluster 2 (FP). Esto significa que *SVM* y *RF* han aprendido a reconocer dos clusters: uno casi inmóvil y otro con una velocidad en z v_z distinta a cero, y por ende en el paso inicial clasifican todo el blanco como cluster 1 y todo el proyectil como cluster 2. La misma clasificación hace *OCSVM*, aunque su aprendizaje es distinto al de *RF* y *SVM*; sólo aprende a reconocer a las partículas que

pertenecerán al cluster 1, y todo lo que sea distinto a este cluster es lo que *OCSVM* etiquetará como cluster 2. Es cierto que dentro del blanco están las partículas que pertenecen al cluster 2, y por ello todas las partículas que pertenecerán al cluster 1 están bien clasificadas (VP da cuenta de la proporción correcta de partículas que pertencerán al cluster 1). Sin embargo, categorizar todo el blanco como el futuro cluster 1 implica una gran cantidad de falsos positivos, que influyen en el cálculo de la precisión:

$$precisión = \frac{1}{1 + \frac{FP}{VP}}, \tag{4.1}$$

haciendo que ésta tenga un valor bajo. A medida que aumentan los pasos temporales y las partículas del blanco empiezan a ser arrastradas por el proyectil, adquiriendo sus velocidades finales, la tasa de FP respecto de VP disminuye, deviniendo en un aumento de precisión.

Por otro lado, como todos los elementos del proyectil terminarán siempre en el cluster 2, estos algoritmos no yerran en esta clasificación y por ende el valor de FN generalmente es nulo en todos los pasos temporales. En general, el algoritmo va cambiando la clasificación del cluster 2 según la evolución del sistema: primero, todo el proyectil es identificado como cluster 2 debido a su velocidad inicial distinta de cero. A medida que este impacta y transfiere momento lineal a las partículas del blanco, el algoritmo va agregando estas partículas del blanco a la categoría de cluster 2, ya que adquieren velocidad en z. Es un caso atípico que una partícula del blanco con velocidad en z adquirida por la colisión frene y permanezca en lo que será el cluster 1. Por lo tanto, su clasificación como cluster 2 será acertada y la tasa de FN será nula en cada paso temporal. Sin embargo pueden ocurrir casos en los cuales el valor de FN no sea exactamente cero en todos los pasos. Si sucede que la partícula disminuye tal velocidad adquirida por el impacto hasta valores cercanos a cero y por ende resulta pertenecer al cluster 1, entonces en algún paso será mal clasificada como cluster 2 y la tasa de FN del predictor no será nula, pero sí muy cercana a cero (las partículas que experimenten tal fenómeno no serán muchas debido a la poca probabilidad de esta situación en colisiones de baja velocidad como las trabajadas).

Otra razón por la cual los FN no serían nulos es el caso de un sobreajuste por parte del predictor del algoritmo, el cual puede clasificar mal en los pasos finales sobre el cluster 1, en donde partículas etiquetadas previamente como cluster 1 en forma correcta sean catalogadas como parte del cluster 2 (disminuyen los VP), haciendo que la tasa de FN aumente un poco (partículas mal etiquetadas como cluster 2), como se verá más adelante. Sin embargo, normalmente se tiene un valor casi nulo de FN y esto, en el cálculo de la métrica sensibilidad, implica siempre valores altos de la misma:

$$sensibilidad = \frac{1}{1 + \frac{FN}{VP}}. \tag{4.2}$$

Este comportamiento de los valores de la MC se repetirá en todos los casos de evolución correcta de las métricas al evaluar los algoritmos.

Se realizó otra prueba general para observar la habilidad de aprendizaje de los algoritmos al entrenarlos con todas las configuraciones finales de las 15 simulaciones de entrenamiento, y luego evaluando cada algoritmo en cada uno de los 15 casos de tamaño pequeño. El conjunto de entrenamiento $\mathcal{S}$ que se les da a los algoritmos se encuentra esquematizado en la Fig. 4.4:

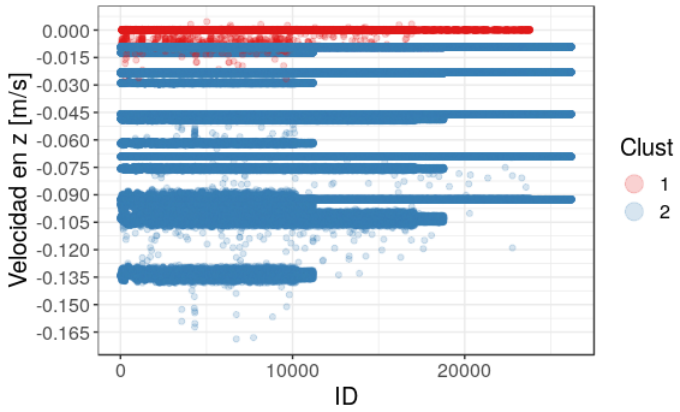

Figura 4.4.: Se muestra la variable predictora $\vec{v_z}$ en función del ID de las partículas para el conjunto de entrenamiento $\mathcal{S}$: todas las partículas de las configuraciones finales en el último paso temporal de todos los casos de entrenamiento, coloreadas por pertenencia a cluster. Cada agrupación horizontal, en azul, corresponde al cluster 2 de cada caso, con las velocidades en z mostradas en la Fig. 3.5.

Como se puede observar en la Fig. 4.4, existe una superposición de clasificaciones de cluster 1 y cluster 2 en el conjunto de entrenamiento. En el intervalo $0\frac{m}{s}\vec{v_z} - 0.015\frac{m}{s}$ hay una gran cantidad de partículas cuyas velocidades en z para un caso pertenecen al cluster 1, mientras que para otro caso, las mismas partículas pertenecen al cluster 2. A pesar de que la separación entre centroides de clusters para cada caso individual está bien delimitada, al yuxtaponer casos distintos, esta separación no es lo suficientemente diferente como para evitar superponer las clasificaciones. A esta franja del entrenamiento se la denomina *franja de superposición*. Los elementos de este entrenamiento $\mathcal{S}$ no son separables linealmente y por lo tanto, *SVM* se entrenó con un núcleo radial.

En general, las métricas de desempeño de los tres algoritmos evolucionaron de forma correcta de forma similar a la Fig. 4.2, exceptuando algunos comportamientos anómalos en la exactitud de *SVM* y *RF*, como el mostrado en la Fig. 4.5. Los casos en los cuales ocurrió este tipo de comportamiento fueron los tres casos $(0.1\frac{m}{s}, \phi)$, y $(0.25\frac{m}{s}, 0.25)$ y $(0.25\frac{m}{s}, 0.35)$.

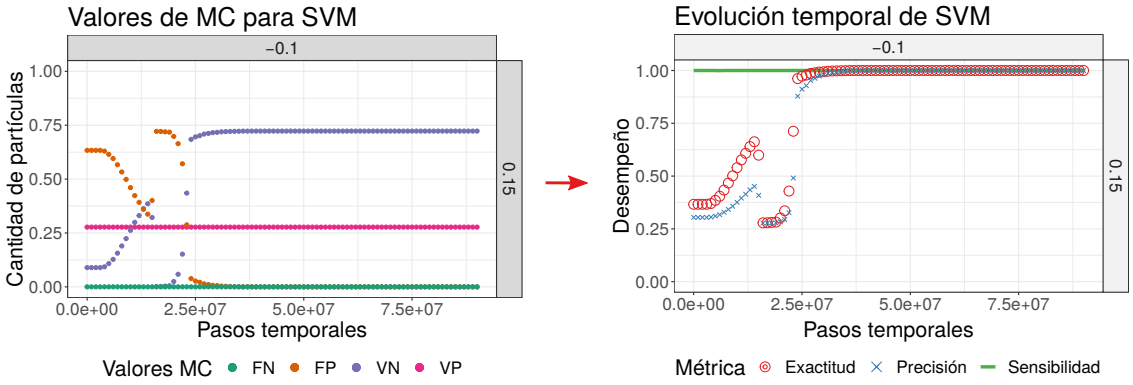

Figura 4.5.: Ejemplo de evolución anómala de la exactitud en *SVM*, para el caso $(0.1\frac{m}{s}, 0.15)$ (der.), con sus respectivos valores de la Matriz de Confusión MC normalizados (izq.). Notar que los pasos temporales están en escala lineal.

Tanto *SVM* como *RF* tuvieron problemas del tipo mostrado en la Fig. 4.5 para la velocidad $\vec{v_0} = -0.1\frac{m}{s}$. A pesar de no describir una curva suave en pasos intermedios de simulación, ambos algoritmos logran tener la exactitud máxima en los pasos finales.

De los valores matriciales se observa que existe un grupo de pasos (aproximadamente entre los 1.5e+07 y 2.5e+07) en los cuales la tasa de partículas mal clasificadas como que pertenecerán al cluster 1 (FP) aumenta abruptamente, para luego decaer rápidamente a valores casi nulos. En este mismo intervalo, la tasa de partículas correctamente clasificadas como futuro cluster 2 (VN) tiene el comportamiento contrario, primero decae y luego aumenta hasta su valor final. El primer fenómeno impacta en el valor de la precisión, haciendo que se comporte de la forma mostrada en el gráfico de la derecha de la Fig. 4.5, mientras que el valor anómalo de los VN impacta en la curva de la exactitud, al ser su ecuación

$$Exactitud = \frac{VP + \mathbf{VN}}{VP + VN + FP + FN}. \tag{4.3}$$

Este comportamiento se debe a que para $\vec{v_0} = -0.1\frac{m}{s}$, la cota máxima de velocidad en z V que pueden alcanzar las partículas del cluster 2 dadas por la Ec. 3.1 es menor en módulo a $0.015\frac{m}{s}$ (Fig. 3.5) y están dentro de la franja de superposición del entrenamiento (Fig. 4.4). Las partículas del cluster 2 dentro de este intervalo corresponden a una minoría si se compara con la cantidad de partículas que pertenecen al resto de los cluster 2. Debido al gran rango de velocidades finales en z que puede adquirir este cluster según el entrenamiento $\mathcal{S}$, el truco de núcleo utilizado por *SVM* para separar lo datos no logra generalizar correctamente sus predicciones a velocidades finales de cluster 2 mayores a $\vec{v_z} = -0.015\frac{m}{s}$, puesto que los casos de velocidades v_0 mayores tienen más ejemplares en este cluster, produciendo un sesgo que influye en la clasificación. En el caso de *RF*, la mala clasificación se debe a que el predictor tendrá opiniones finales con el mismo peso para clasificar una partícula del intervalo de superposición como cluster 1 o 2. A pesar de la mala clasificación en los pasos intermedios, como los predictores alcanzan el valor de exactitud máximo, podría tomarse el resultado como un buen desempeño, aunque si se interpreta el comportamiento anómalo como producto de un sobreajuste del predictor en ciertos pasos, no sería este el escenario óptimo buscado.

OCSVM, en cambio, simplemente reconoce todos los puntos que pertenecen al cluster 1, y como posee todos los ejemplos posibles en su conjunto de entrenamiento, reconoce (casi) todos los puntos, resultando en un desempeño correcto en todos los casos. La exactitud para *OCSVM*, sin embargo, nunca llega a 1, a diferencia de las de *SVM* y *RF*, puesto que *OCSVM* clasifica en todos los casos como que pertenecerán al cluster 1 a las pocas partículas superpuestas del cluster 2 que caigan en la región de la superposición de clasificaciones.

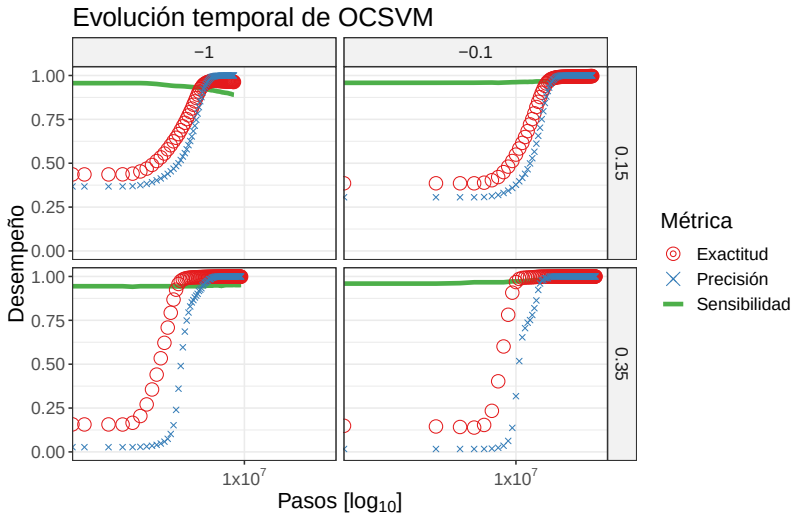

Figura 4.6.: Evolución de las métricas para los cuatro casos extremos de *OCSVM*. En las columnas se muestran dos $\vec{v_0}$ y en las filas, dos valores de ϕ. La evolución del resto de los casos es semejante.

Notar que en el caso $(1\frac{m}{s}, 0.15)$ tiene una pequeña disminución en la métrica de la sensibilidad, debido a un aumento en los valores de partículas mal clasificadas como cluster 2 (FN). Este aumento se compensa con una disminución en la clasificación correcta de partículas que pertenecerán al cluster 1 (VP). Esto es indicio de un pequeño sobreajuste por parte de la función predictora de *OCSVM*, pero su influencia sobre la exactitud es casi imperceptible.

Conclusión de pruebas preliminares

Ninguna de las dos pruebas realizadas en este sección, a pesar de tener casos con evolución correcta de las métricas, es útil para mejorar el tiempo de simulación. En las dos, es necesario tener las 15 simulaciones de entrenamiento completas hasta el final, para poder etiquetar y entrenar con el último paso de cada una. Debido a esto, el porcentaje de ganancia en ambas es nulo. Estas pruebas preliminares simplemente son un control de desempeño: se observó el comportamiento de las métricas de cada algoritmo, el cual concuerda con las características de los algoritmos entrenados.

Debido al desempeño no tan satisfactorio de *SVM* y *RF* cuando se los entrenó con $\mathcal{S}$, los resultados apoyan la elección de *OCSVM* como el algoritmo más adecuado para esta tarea de clasificación, debido a su evolución correcta de exactitud aún a pesar de poseer un conjunto de entrenamiento linealmente no separable.

4.3 Extensión local

Esta sección comprende pruebas con distintos conjuntos de entrenamiento $\mathcal{S}_*$, donde todos los datos etiquetados se toman de los archivos de salida correspondientes a los pasos de finalización de las simulaciones de entrenamiento. Llamaremos $T_{n_{peq}}$ al T_n que aparece en la fórmula 3.10; para todas las pruebas de extensión local su valor es de $T_{n_{peq}}$ = 15d 23H 19M (días, horas, minutos). El tiempo ahorrado debido a las predicciones de los algoritmos en cada prueba es respecto a este tiempo $T_{n_{peq}}$.

4.3.1 Pruebas por ϕ

Se realizaron tres pruebas con entrenamientos formados por las configuraciones finales de las simulaciones de entrenamiento agrupadas por ϕ en todas las velocidades v_0 y testeando en cada caso, es decir, se tienen tres conjuntos de entrenamiento distintos: $\mathcal{S}_{\phi=0.15}$, $\mathcal{S}_{\phi=0.25}$ y $\mathcal{S}_{\phi=0.35}$ (Fig. 4.1). En la Fig. 4.7 se esquematizan los tres entrenamientos utilizados en esta sección, es decir, los elementos del dominio

(todas las partículas constituyentes del último paso de las simulaciones de tamaño pequeño, agrupadas por ϕ) etiquetados según pertenezcan al cluster 1 o 2 luego de la colisión (color rojo o azul, respectivamente). Por las características de las partículas correspondientes a cada cluster y por los resultados mostrados en la sección anterior, se estima a priori que debido a la superposición en los casos de factor de llenado $\phi = 0.15$ y 0.25, no parecen ser estos los más adecuados para entrenar a los algoritmos *SVM* y *RF*. Para *SVM* se utilizó nuevamente un núcleo radial en el entrenamiento de estos dos valores de ϕ, debido a que los datos no son separables linealmente.

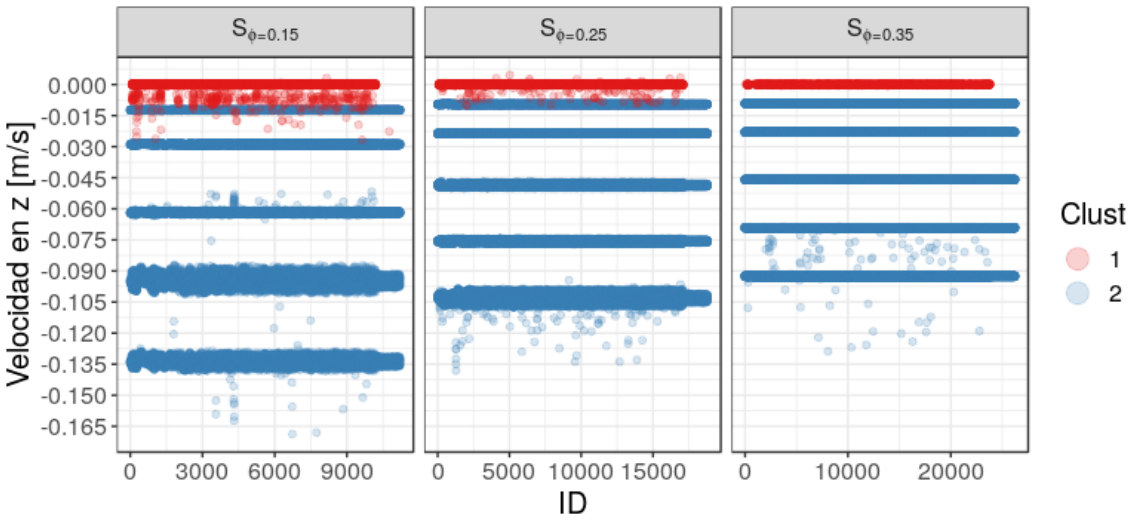

Figura 4.7.: Se muestra la variable predictora $\vec{v_z}$ en función de el ID de las partículas para los tres conjuntos de entrenamiento $\mathcal{S}_{\phi=0.15}$, $\mathcal{S}_{\phi=0.25}$ y $\mathcal{S}_{\phi=0.35}$. Cada uno de los 5 grupos horizontales distinguibles de partículas (en azul) representa las velocidades finales en z de las partículas del cluster 2 para cada v_0, (Fig. 3.5).

Para $\mathcal{S}_{\phi=0.25}$, la franja de superposición es menos densa, y la velocidad final en z de las partículas en donde hay más superposición es alrededor de los $-0.01\frac{m}{s}$, mientras que para $\mathcal{S}_{\phi=0.15}$, la franja de superposición es más ancha y la velocidad final en z donde hay más superposición es cerca de los $-0.02\frac{m}{s}$.

Resultados de $\mathcal{S}_{\phi=0.15}$

Tal como se predijo al observar los entrenamientos, la evolución temporal de las métricas de *SVM* y *RF* no fue correcta en cuatro casos, esquematizados en la Fig. 4.8. La exactitud nunca llega a 1, y los casos donde esto sucede son los de mayor ϕ y menor v_0, que son justamente aquellos en la zona de superposición de datos en $\mathcal{S}_{\phi=0.15}$.

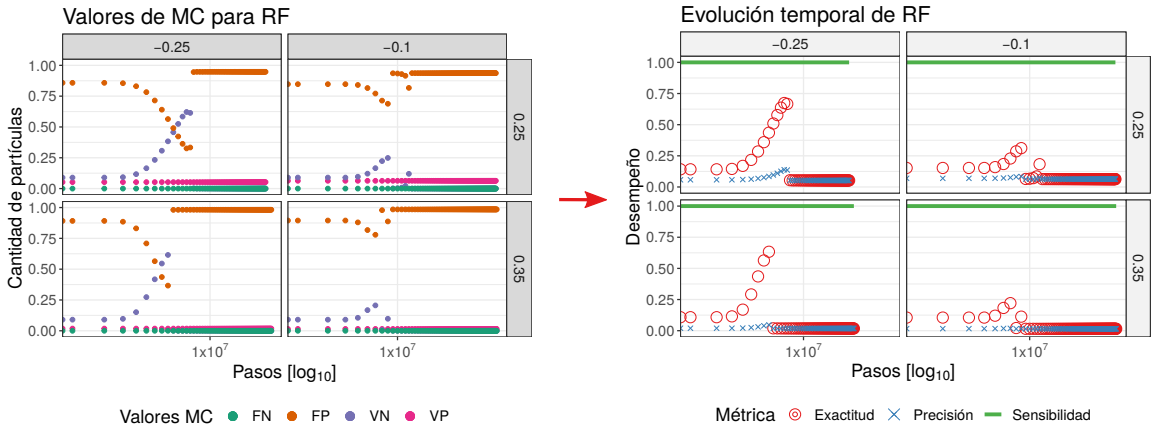

Figura 4.8.: Valores de la MC en función de los pasos temporales (izq.) para casos de evoluciones no correctas de *RF* (der.). Los gráficos están divididos por $\vec{v_0}$ (columnas) y ϕ (filas). Notar que los pasos temporales están en escala logarítmica.

La Fig. 4.8 muestra que para los cuatro casos $(0.25\frac{m}{s}, 0.25)$, $(0.1\frac{m}{s}, 0.25)$, $(0.25\frac{m}{s}, 0.35)$, $(0.1\frac{m}{s}, 0.35)$, *RF* inicia en los primeros pasos la clasificación de forma correcta, con la exactitud y la precisión aumentando paulatinamente, repitiendo el patrón mostrado en la Fig. 4.3. Sin embargo, al llegar a un cierto paso, el predictor de *RF* comienza abruptamente a clasificar todas las partículas del sistema como que pertenecerán al cluster 1, y por lo tanto la tasa de VN (partículas correctamente clasificadas como futuro cluster 2) también decaen drásticamente. Esto implica que tanto la exactitud como la precisión disminuyen sus valores hasta casi cero. La sensibilidad, sin embargo, sigue teniendo un valor máximo debido a que siempre se clasifican bien las partículas del cluster 1.

Si se toma como ejemplo el caso $(0.25\frac{m}{s}, 0.25)$, se observa que se produce este decaimiento de la exactitud cuando las partículas del proyectil, luego de impactar totalmente en el blanco, desaceleran hasta una velocidad de aproximadamente $-0.02\frac{m}{s}$ (cayendo dentro de la región de superposición de etiquetas mostrada por la Fig. 4.7), y las partículas del blanco que comienzan a ser arrastradas por él también adquieren esta velocidad (Fig. 4.9). Es evidente que el clasificador de RF toma las partículas que poseen su v_z dentro de la zona de superposición del entrenamiento como pertenecientes al cluster 1. Para los cuatro casos de la Fig. 4.8 el decaimiento en la exactitud se produce en esta etapa de la colisión; cuando el proyectil y las partículas que serán arrastradas por él adquieren sus velocidades finales en z.

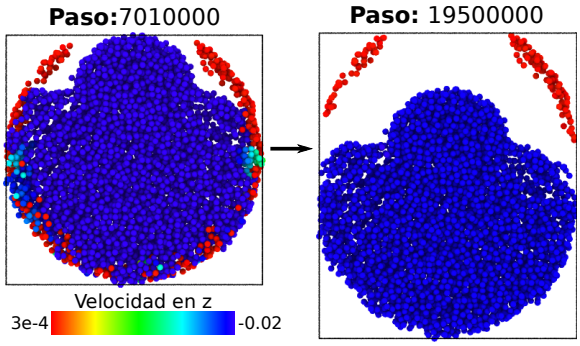

Figura 4.9.: Ilustración del paso (7010000) en el cual se produce la mala clasificación de RF con el entrenamiento $\mathcal{S}_{\phi=0.15}$ en el caso $(0.25\frac{m}{s}, 0.25)$. El proyectil y las partículas que éste arrastra en la colisión (futuro cluster 2) adquieren una velocidad de aproximadamente $-0.02\frac{m}{s}$ (color naranja), que se mantiene luego de la separación de los clusters. Se muestra una rodaja de $1e-05m$ de espesor paralela al eje z de la colisión.

Notar que el algoritmo clasifica bien los casos de $\phi = 0.15$, lo cual indica que aprendió correctamente a clasificar los casos de provistos como entrenamiento.

Los casos problemáticos de SVM fueron parecidos a los de RF y de la misma forma. Por otro lado, $OCSVM$ tuvo un desempeño correcto para los 15 casos, es decir, mediante el entrenamiento de los cinco casos con $\phi = 0.15$ se puede predecir el desenlace de las otras diez simulaciones. Nuevamente se produce un pequeño sobreajuste en los últimos pasos de los casos $(0.75\frac{m}{s}, 0.15)$, $(1\frac{m}{s}, 0.15)$ y $(0.5\frac{m}{s}, 0.25)$, manifestado en la curvatura hacia abajo en la evolución de la exactitud producto del aumento de

la tasa de FN, impidiendo que el predictor alcance el 99 % de exactitud en este caso. Sin embargo, en los tres casos el predictor alcanza un 97 % de exactitud antes de decaer, por lo cual se considera en general un buen desempeño de *OCSVM*.

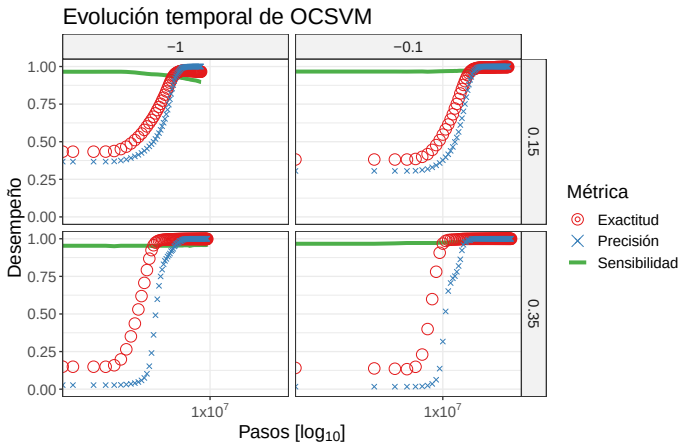

Figura 4.10.: Evolución de las métricas en función de los pasos temporales para cuatro casos extremos de *OCSVM* entrenado con $\mathcal{S}_{\phi=0.15}$. La evolución del resto de los casos es semejante. Los casos se encuentran graficados por $\vec{v_0}$ (columnas) y ϕ (filas).

El ahorro temporal y el porcentaje de ganancia obtenidos de la utilización de *OCSVM* se encuentran especificados en la tabla 4.1, donde se observan los valores de tiempo especificados en la Secc. 3.5. Los valores faltantes del 99 % indican el hecho de que *OCSVM* no alcanzó este valor en todos los casos debido al sobreajuste del que se ha hablado anteriormente.

$T_{n_{peq}}$	T_e	% Ex.	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	4d 8H 45M	90 %	6d 1H 49M	9d 22H 30M	62.2 %
		95 %	6d 5H 13M	9d 18H 6M	61.1 %
		99 %	-	-	-

Tabla 4.1: Tabla de costos y ahorros temporales de las predicciones de *OCSVM* entrenado con $\mathcal{S}_{\phi=0.15}$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_{\phi=0.15}$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}$−T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

Resultados de $\mathcal{S}_{\phi=0.25}$

Para este conjunto de entrenamiento, los algoritmos *OCSVM* y *SVM* tuvieron buen desempeño (Fig. 4.11), con un comportamiento de la sensibilidad y de precisión semejante al de las pruebas preliminares. Comparando los entrenamientos $\mathcal{S}_{\phi=0.15}$ y $\mathcal{S}_{\phi=0.25}$ (Fig. 4.7), el último tiene menos datos superpuestos y por ello *SVM* puede linealizarlos mediante una transformación de manera más eficiente.

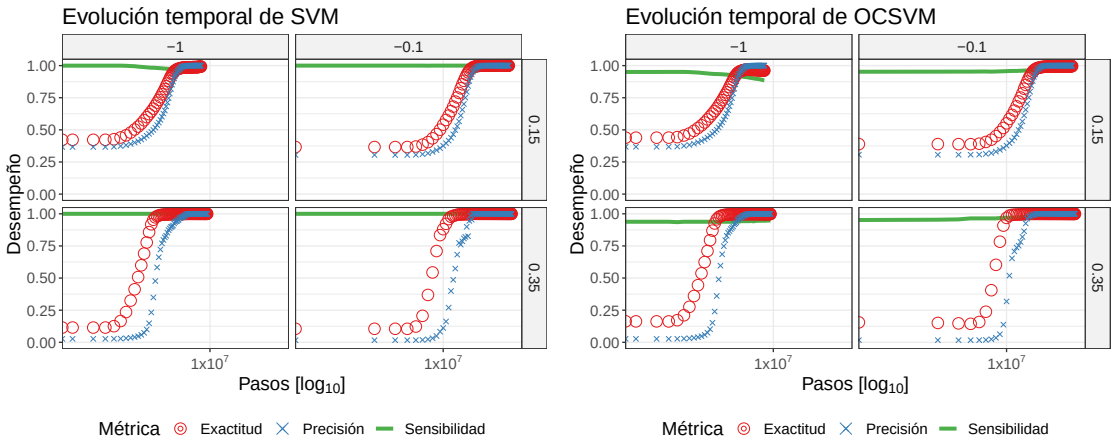

Figura 4.11.: Evolución de las métricas en función de los pasos temporales de los casos extremos para *SVM* y *OCSVM*. Los casos se grafican por $\vec{v_0}$ (columnas) y ϕ (filas).

RF sin embargo, falla para los casos $(0.1\frac{m}{s}, 0.15)$ y $(0.1\frac{m}{s}, 0.35)$, como se muestra en la Fig. 4.12. Notar que el algoritmo falla en menos casos que cuando fue entrenado con $\mathcal{S}_{\phi=0.15}$, y no son exactamente los mismos. Que *RF* haya sido capaz de clasificar bien los casos de $\phi = 0.25$ implica que hubo un buen aprendizaje de los casos que se le dieron como entrenamiento.

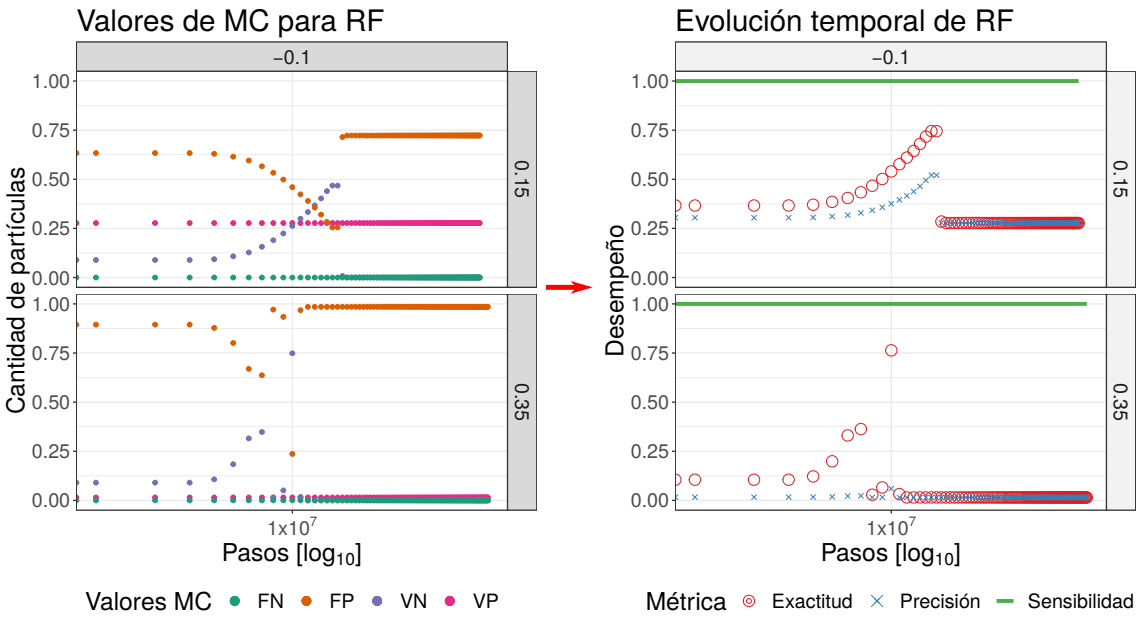

Figura 4.12.: Valores de la MC en función de los pasos temporales (izq.) para casos de evoluciones no correctas de *RF* (der.). Los gráficos están divididos por $\vec{v_0}$ (columnas) y ϕ (filas). Notar que los pasos temporales están en escala logarítmica.

A diferencia de *RF*, *SVM* y *OCSVM* han logrado un desempeño correcto en las 15 simulaciones evaluadas, lo cual implica que han logrado generalizar sus predicciones a todo el conjunto de simulaciones, no sólo a las de su entrenamiento. El tiempo de ganancia de simulación G_t logrado por *SVM* y *OCSVM* se resume en la siguiente tabla:

$T_{n_{peq}}$	T_e	% Ex.	Algoritmo	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	6d 1H 18M	90 %	*SVM*	7d 21H 13M	8d 2H 5M	50.6 %
			OCSVM	7d 20H 19M	8d 3H 12M	50.9 %
		95 %	*SVM*	8d 1H 32M	7d 21H 46M	49.5 %
			OCSVM	8d 1H 27M	7d 21H 51M	49.5 %
		99 %	*SVM*	8d 9H 59M	7d 13H 20M	47.3 %
			OCSVM	-	-	-

Tabla 4.2: Tabla de costos y ahorros temporales de *SVM* y *OCSVM* entrenado con $\mathcal{S}_{\phi=0.25}$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_{\phi=0.25}$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

Cabe destacar que *SVM* llega a una exactitud del 99 % en todos los casos, mientras que *OCSVM* en los casos ($0.5\frac{m}{s}$, 0.15), ($0.75\frac{m}{s}$, 0.15) y ($1\frac{m}{s}$, 0.15) sólo alcanza un 97 % de exactitud antes de decaer debido a un pequeño sobreajuste.

Resultados de $\mathcal{S}_{\phi=0.35}$

Con este entrenamiento los tres algoritmos se desempeñan correctamente en los 15 casos. Los resultados de algunos casos representativos de dos algoritmos se muestran en la Fig. 4.13, pues el resto son muy similares.

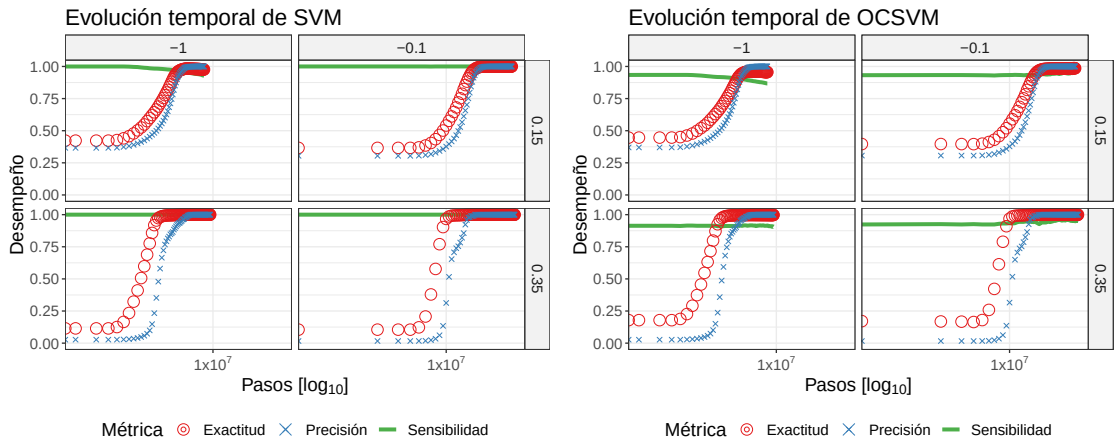

Figura 4.13.: Evolución de las métricas de los casos extremos para *SVM* y *OCSVM*. *RF* es muy parecido a *SVM*. Los casos se grafican por $\vec{v_0}$ (columnas) y ϕ (filas).

El caso $(1\frac{m}{s}, 0.15)$ presenta una pequeña disminución hacia el final de la sensibilidad, más pronunciada para *OCSVM* que para *RF* y *SVM*. Si observamos los valores de la Matriz de Confusión MC de este caso para *SVM* y *OCSVM* (Fig. 4.14) hay una disminución de VP más marcada en *OCSVM*, lo cual implica un crecimiento en las partículas mal clasificadas como que pertenecerán al cluster 2 (FN). Sin embargo, el decrecimiento de los VP se compensa con el crecimiento de VN y la exactitud no fluctúa por este pequeño sobreajuste.

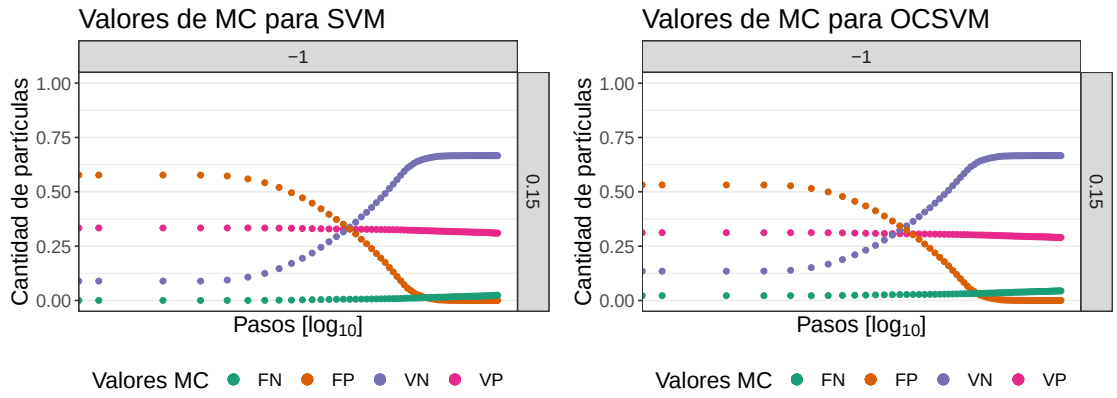

Figura 4.14.: Comparación de los valores de la MC de *SVM* y *OCSVM* en función de los pasos temporales (izq.) para el caso $(1\frac{m}{s}, 0.15)$. Notar que los pasos temporales están en escala logarítmica.

Como los tres algoritmos logran desempeñarse bien en los 15 casos evaluados, es decir, logran generalizar las etiquetas del entrenamiento a otros casos de forma correcta, se logra un porcentaje de ganancia G_t para todos los casos, detallados en la siguiente tabla:

$T_{n_{peq}}$	T_e	% Ex.	Algoritmo	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	5d 13H 15M	90 %	*SVM*	7d 19H 10M	8d 4H 8M	51.2 %
			RF	7d 19H 10M	8d 4H 8M	51.2 %
			OCSVM	7d 19H 20M	8d 3H 59M	51.1 %
		95 %	*SVM*	8d 0H 4M	7d 23H 14M	49.9 %
			RF	8d 0H 9M	7d 23H 9M	49.9 %
			OCSVM	8d 1H 24M	7d 21H 54M	49.5 %
		99 %	*SVM*	-	-	-
			RF	-	-	-
			OCSVM	-	-	-

Tabla 4.3: Tabla de costos y ahorros temporales de *SVM*, *RF* y *OCSVM* entrenado con $\mathcal{S}_{\phi=0.35}$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_{\phi=0.35}$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

Ninguno de los algoritmos logra alcanzar una exactitud del 99 % para todos los casos evaluados, debido a un sobreajuste del predictor. *OCSVM* sólo alcanzan entre el 96 % y 97 % en todos los casos de $\phi = 0.15$, y en el caso $(0.75\frac{m}{s}, 0.25)$, en cambio, *SVM* y *RF* alcanzan el 98 % de exactitud en el caso $(0.75\frac{m}{s}, 0.15)$.

Conclusiones de pruebas por ϕ

En la Fig. 4.15 se muestran los porcentajes de ganancias temporales G_t especificados en las tablas 4.1, 4.2 y 4.3 de los tres entrenamientos $\mathcal{S}_{\phi=0.15}$, $\mathcal{S}_{\phi=0.25}$ y $\mathcal{S}_{\phi=0.35}$.

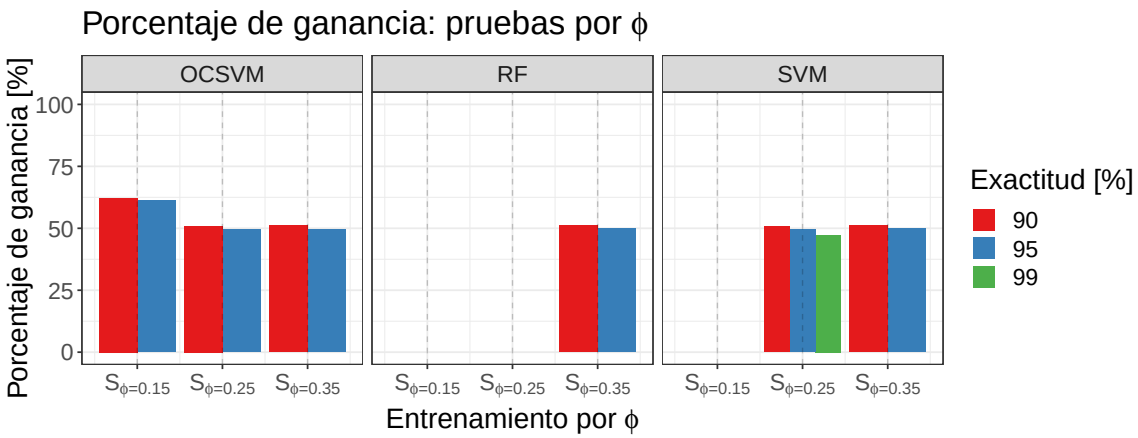

Figura 4.15.: Porcentaje de ganancia G_t para los tres entrenamientos por ϕ.

Solo *SVM* llega al 99 % de exactitud para todos los casos con el entrenamiento $\mathcal{S}_{\phi=0.25}$, y cabe destacar que no hay mucha diferencia de porcentaje de ganancia entre exactitudes para ningún entrenamiento, por lo tanto se recomienda que el analista ejecute las simulaciones hasta al menos el 95 % de exactitud: las predicciones serán más precisas (más confiables), y no más costosas en términos de tiempo. Además, en las pruebas en las cuales *RF* o *SVM* logran un buen desempeño, en general tienen más casos evaluados que llegan al 99 % de exactitud que *OCSVM*. Notamos que el ahorro mayor para los valores de exactitud sugeridos lo tiene *OCSVM* con el entrenamiento de $\mathcal{S}_{\phi=0.15}$, de al rededor del 60 %. El resto de las ganancias rondan el 50 %.

Los tres algoritmos pueden generalizar el aprendizaje al resto de los datos con el conjunto de entrenamiento cuyos datos etiquetados están mejor delimitados: $\mathcal{S}_{\phi=0.35}$. La desventaja de este entrenamiento es que sus simulaciones son las que más tiempo real demoran en obtenerse porque son los casos que más partículas tienen.

4.3.2 Pruebas por v_0

Se prepararon cinco conjuntos de entrenamiento, correspondientes a las configuraciones finales de cada una de las simulaciones de entrenamiento agrupadas por velocidad inicial de proyectil v_0 (Fig. 4.1): $\mathcal{S}_{v_0=0.1}$, $\mathcal{S}_{v_0=0.25}$, $\mathcal{S}_{v_0=0.5}$, $\mathcal{S}_{v_0=0.75}$ y $\mathcal{S}_{v_0=1}$.

Cada uno de estos entrenamientos contiene los tres casos de ϕ correspondientes a cada v_0, y se evaluó sobre cada uno de los 15 casos de tamaño pequeño. La forma de los conjuntos de entrenamiento se muestra en la Fig. 4.16. Los datos de los cinco entrenamientos son linealmente separables y por ende se entrenó *SVM* con núcleo lineal.

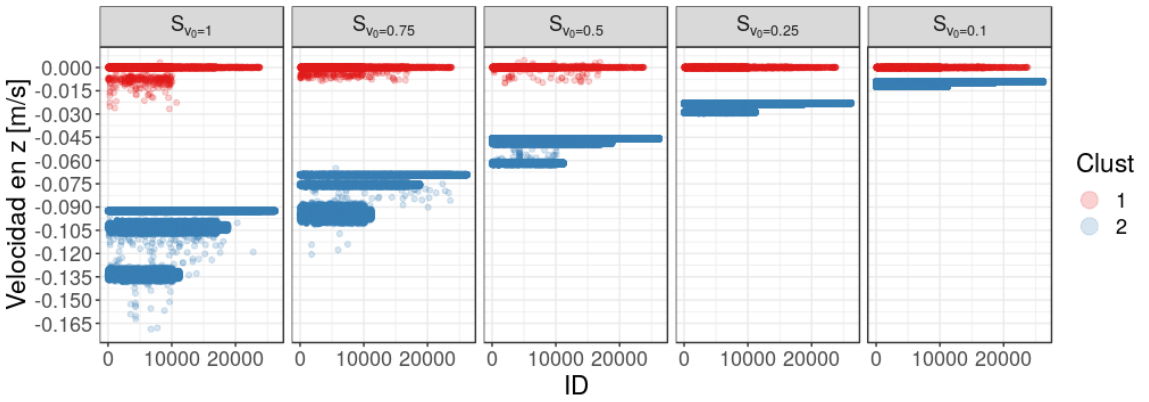

Figura 4.16.: Se muestra la variable predictora $\vec{v_z}$ en función de el ID de las partículas para los cinco conjuntos de entrenamiento $\mathcal{S}_{v_0=0.1}$, $\mathcal{S}_{v_0=0.25}$, $\mathcal{S}_{v_0=0.5}$, $\mathcal{S}_{v_0=0.75}$ y $\mathcal{S}_{v_0=1}$ coloreadas por pertenencia a cluster. Los tres grupos horizontales de partículas azules distinguibles en cada entrenamiento reflejan las velocidades finales en z del cluster 2 para los tres factores de llenado, siendo $\phi = 0.15$ la franja inferior, $\phi = 0.25$ la del medio y $\phi = 0.35$ la superior (Fig. 3.5).

Como muestra la Fig. 4.16, para los cinco casos se distinguen tres agrupaciones de clusters, correspondientes a los casos de (v_0, 0.15), (v_0, 0.25), (v_0, 0.35). Esta diferencia de velocidades finales en z de los tres ϕ es debido a la cota que poseen en sus velocidades V, calculadas mediante la Ec. 3.1: $V = \frac{m_p}{m_b+m_p} v_0$ (Fig. 3.5). En cada entrenamiento, v_0 no cambia, pero sí lo hace la cantidad de partículas que arrastra el proyectil. Como hay más partículas en el mismo volumen para $\phi = 0.35$, el proyectil experimentará una fuerza de frenado efectiva, por conservación de momento lineal, mucho mayor que los otros ϕ, y por lo tanto el cluster 2 adquirirá una magnitud de v_z final menor a los clusters 2 de los casos (v_0, 0.15) y (v_0, 0.25). Notar que para velocidades menores, las separación (es decir, la diferencia de velocidad) entre los cluster 1 y 2 es cada vez menor.

Ninguno de estos entrenamientos tiene problemas de superposición de etiquetas de clusters distintos, pero sí un problema relacionado con la separación entre centroides de cada cluster. No se puede esperar que el conjunto de entrenamiento $\mathcal{S}_{v_0=1}$ caracterice bien los casos que corresponden a ($v_0 = 0.1\frac{m}{s}$, ϕ), puesto que la configuración en el último paso en estos casos tienen los centroides de los clusters demasiado cercanos en comparación con los del conjunto de entrenamiento. Adicionalmente, los valores de la velocidad en z del cluster 2 están en el mismo intervalo que los valores de v_z que corresponden al cluster 1 para $\mathcal{S}_{v_0=1}$. Así, no se espera una evolución correcta para ninguno de los casos que entrenen configuraciones finales de clusters demasiado separados sobre casos cuyas configuraciones finales tengan centroides demasiado cercanos.

Resultados de $\mathcal{S}_{v_0=0.1}$

La exactitud en los 15 casos para los tres algoritmos evoluciona de manera correcta. Se estima que el buen comportamiento de esta métrica se debe a que el conjunto de entrenamiento $\mathcal{S}_{v_0=0.1}$ (Fig. 4.16) tiene sus centroides muy cercanos y bien delimitados. Ningún caso tendrá en su configuración final partículas pertenecientes al cluster 2 con magnitud v_z menor que las del cluster 2 del entrenamiento (Fig. 3.5), y por ende los algoritmos reconocen bien este grupo en el resto de los casos con $v_0 \neq 0.1\frac{m}{s}$. A continuación se muestran cuatro casos representativos de la evolución de los algoritmos.

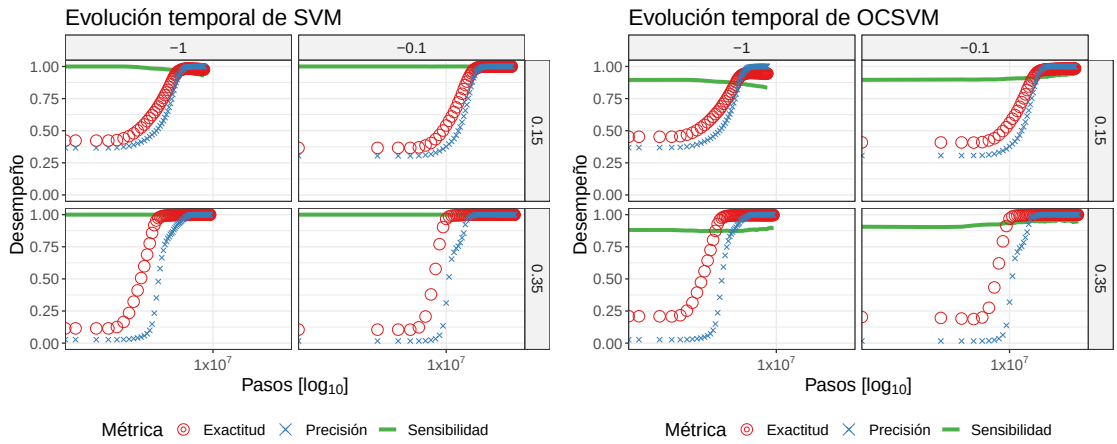

Figura 4.17.: Evolución de las métricas de los cuatro casos extremos para *SVM* y *OCSVM*. *RF* tiene un desempeño muy parecido a *SVM*. Los casos se grafican por $\vec{v_0}$ (columnas) y ϕ (filas).

En la Fig. 4.17 se observa que para *OCSVM* y *SVM* (y *RF*, el cual se comporta de manera semejante a *SVM*) el caso $(1\frac{m}{s}, 0.15)$ tiene un decaimiento en la sensibilidad. Esto se había observado anteriormente en las pruebas por ϕ, es a causa de una disminución en la tasa de VP respecto de FN, más marcada en las predicciones de *OCSVM*. Nuevamente, este pequeño sobreajuste no influye marcadamente en la evolución de la exactitud.

Como los tres algoritmos logran generalizar las etiquetas del entrenamiento por $v_0 = 0.1\frac{m}{s}$ a otros casos de forma correcta, se detalla en la siguiente tabla los valores de las ganancias temporales G_t:

$T_{n_{peq}}$	T_e	% Ex.	Algoritmo	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	8d 18H 7M	90 %	*SVM*	10d 5H 53M	5d 17H 25M	35.8 %
			RF	10d 5H 53M	5d 17H 25M	35.8 %
			OCSVM	10d 6H 27M	5d 16H 51M	35.7 %
		95 %	*SVM*	10d 9H 22M	5d 13H 56M	34.9 %
			RF	10d 9H 22M	5d 13H 56M	34.9 %
			OCSVM	10d 9H 57M	5d 13H 21M	34.8 %
		99 %	*SVM*	-	-	-
			RF	-	-	-
			OCSVM	-	-	-

Tabla 4.4: Tabla de costos y ahorros temporales de *SVM*, *RF* y *OCSVM* entrenados con $\mathcal{S}_{v_0=0.1}$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_{v_0=0.1}$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

Los casos para los cuales *OCSVM* no alcanza el 99 % de exactitud son todos los de (v_0, 0.15), y ($1\frac{m}{s}$, 0.25) y ($0.75\frac{m}{s}$, 0.25) (alcanzan entre el 96 %-97 %). *RF* y *SVM* logran hasta un 98 % de exactitud en el caso ($1\frac{m}{s}$, 0.15).

Resultados de $\mathcal{S}_{v_0=0.25}$

Para el entrenamiento $\mathcal{S}_{v_0=0.25}$ el desempeño de *SVM* y *RF* sólo fue correcto para los casos de $v_0 \geq 0.25\frac{m}{s}$. Tal como se estimó, con este conjunto de entrenamiento, los casos de velocidad $\vec{v_0} = -0.1\frac{m}{s}$ tuvieron problemas en sus predicciones, lo cual era esperable, puesto que el conjunto de entrenamiento tiene la distancia entre centroides de clusters más alejados que los de la última configuración de la velocidad $\vec{v_0} = -0.1\frac{m}{s}$ y al predictor le cuesta reconocer el cluster 2 de estos casos. Esta hipótesis la confirma el hecho de que *OCSVM* clasifique correctamente todas las simulaciones de evaluación, puesto que sólo necesita reconocer el cluster 1, el cual está bien delimitado (Fig. 4.16).

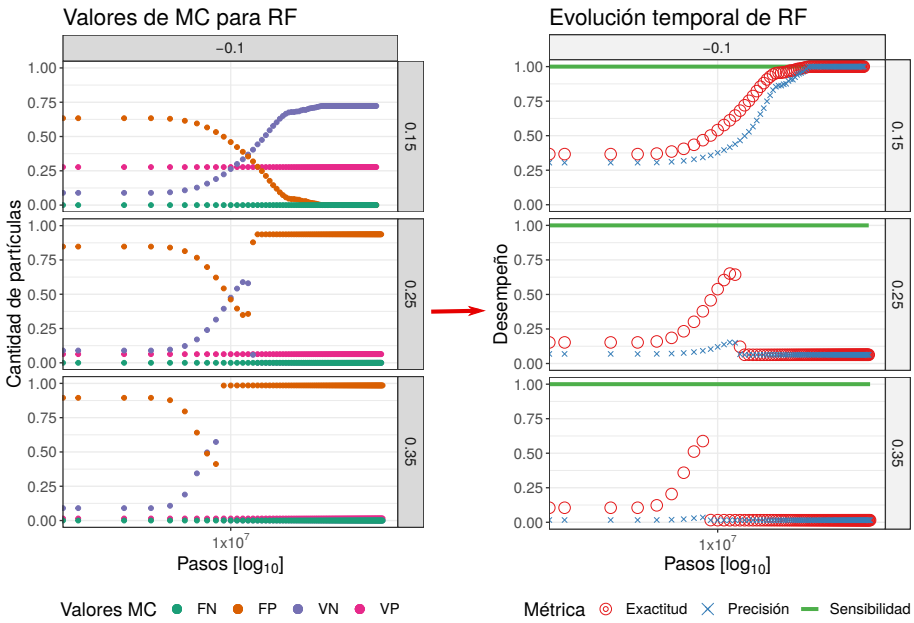

Figura 4.18.: Valores de la MC en función de los pasos temporales (izq.) para casos de evoluciones no correctas de *RF* (der.). *SVM* tiene problemas con los mismos casos, con características similares

. Los gráficos están divididos por $\vec{v_0}$ (columnas) y ϕ (filas). Notar que los pasos temporales están en escala logarítmica.

Se dejó a propósito el caso de $(0.1\frac{m}{s}, 0.15)$, puesto que se esperaba que el algoritmo falle también en él, pero no fue así. Esto se debe a que la velocidad v_z que tiene el cluster 2 de este caso es muy cercano al del entrenamiento usado aquí (Fig. 4.16, Sección 3.5), y por ende los algoritmos lo clasifican sin problemas. Se observa que en ningún caso de mal desempeño la sensibilidad deja de ser 1, puesto que los algoritmos siguen etiquetando como cluster 1 a todas las partículas del blanco. La precisión, sin embargo, es casi nula, al igual que la exactitud, debido al cambio abrupto en la evolución de los valores de FP y VN. Este cambio sucede cuando el proyectil decrece su velocidad inicial al impactar con el blanco, como se explicó en la sección *Resultados de* $\mathcal{S}_{\phi=0.15}$.

El único algoritmo que realiza un ahorro temporal al generalizar bien en todos los casos es *OCSVM*, y se detalla en la siguiente tabla:

$T_{n_{peq}}$	T_e	% Ex.	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	3d 20H 14M	90 %	5d 23H 44M	9d 23H 34M	62.5 %
		95 %	6d 5H 10M	9d 18H 8M	61.1 %
		99 %	-	-	-

Tabla 4.5: Tabla de costos y ahorros temporales de *OCSVM* entrenado con $\mathcal{S}_{v_0=0.25}$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_{v_0=0.25}$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}$−T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

Nuevamente *OCSVM* logra hasta solamente un 96 %-97 % de exactitud en los cinco casos (v_0, 0.15), y ($0.75\frac{m}{s}$, 0.25) debido al sobreajuste que se produce.

Resultados de $\mathcal{S}_{v_0=0.5}$

En esta prueba fallan casi todos los casos con v_0 menor a $0.5\frac{m}{s}$; ni *SVM* ni *RF* pueden distinguir bien el cluster 2 de estos casos con el entrenamiento $\mathcal{S}_{v_0=0.5}$ (Fig. 4.19). Nuevamente se repite la abrupta disminución de la exactitud y de la precisión cuando el proyectil y las futuras partículas del cluster 2 pertenecientes al blanco adquieren sus velocidades finales en z, pues para estas v_0 tan bajas, la velocidad en z disminuye tanto que cae en la zona del entrenamiento en donde el predictor asocia a las partículas con el cluster 1. Todos los casos con malas clasificaciones de la Fig. 4.19 tienen $\vec{v_z}$ finales de cluster 2 mayores a $-0.023\frac{m}{s}$, mientras que el caso ($0.25\frac{m}{s}$, 0.15) tiene una velocidad final de cluster 2 de $-0.029\frac{m}{s}$ y por eso el algoritmo lo clasifica correctamente (ver Fig. 3.5).

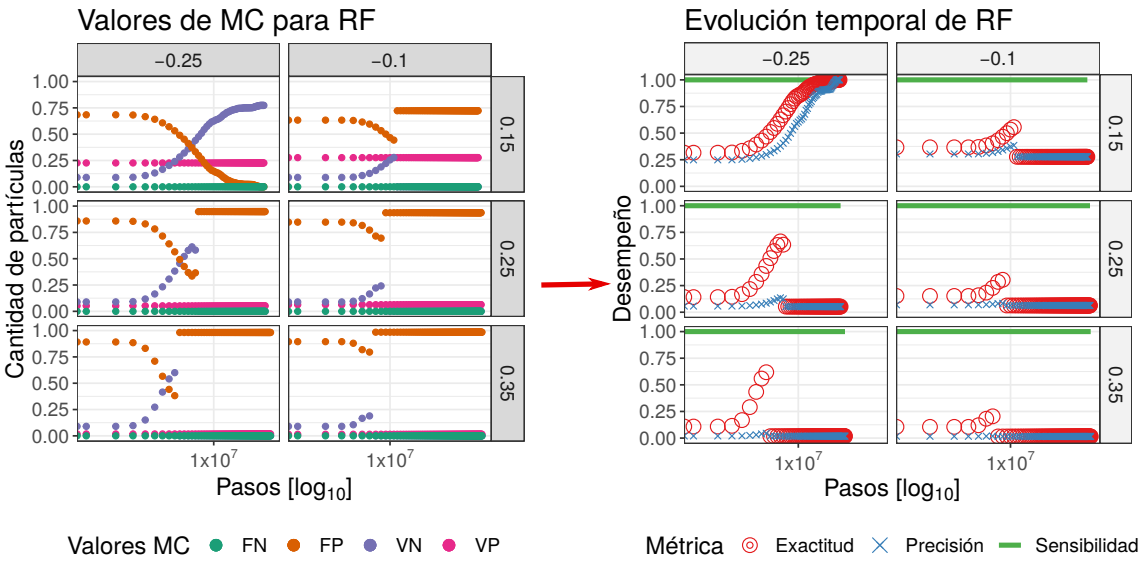

Figura 4.19.: Valores de la MC en función de los pasos temporales (izq.) para casos de evoluciones no correctas de *RF* (der.). Los gráficos están divididos por $\vec{v_0}$ (columnas) y ϕ (filas). Notar que los pasos temporales están en escala logarítmica.

OCSVM, en cambio, nuevamente tiene un buen desempeño en las 15 simulaciones evaluadas, con una evolución semejante a las de pruebas anteriores. El algoritmo logra generalizar el entrenamiento para predecir el desenlace de todas las colisiones tratadas, y el ahorro temporal y porcentaje de ganancia se especifica a continuación:

$T_{n_{peq}}$	T_e	% Ex.	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	1d 9H 19M	90 %	3d 22H 30M	12d 0H 49M	75.3 %
		95 %	4d 5H 17M	11d 18H 2M	73.6 %
		99 %	-	-	-

Tabla 4.6: Tabla de costos y ahorros temporales de *OCSVM* entrenado con $\mathcal{S}_{v_0=0.5}$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_{v_0=0.5}$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}$−T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

A diferencia de las pruebas anteriores en donde alrededor de la mitad de los casos no alcanzaban el 99 % de exactitud, para este entrenamiento solamente no lo logran

los casos ($0.5\frac{m}{s}$, 0.15) ($0.75\frac{m}{s}$, 0.15) y ($1\frac{m}{s}$, 0.15), los cuales alcanzan hasta un 97 % de exactitud.

Resultados de $\mathcal{S}_{v_0=0.75}$

Para este entrenamiento, siguiendo el patrón de comportamiento de los resultados anteriores, se esperaría que *SVM* y *RF* no puedan dar cuenta de los casos de $\vec{v_0} = -0.5\frac{m}{s}$, pero no fue así. Si se observa este conjunto de entrenamiento en la Fig. 4.16, el intervalo de v_z de las partículas del cluster 1 es el mismo que el del entrenamiento $\mathcal{S}_{v_0=0.5}$, aunque hay más densidad de partículas. Debido a esto, los algoritmos clasifican los mismos casos de forma incorrecta. Además, como muestra la Fig. 4.20, $\mathcal{S}_{v_0=0.75}$ posee otro caso mal clasificado, ($0.25\frac{m}{s}$, 0.15), lo cual no sucedió para el entrenamiento $\mathcal{S}_{v_0=0.5}$. Esto puede deberse a que la separación de los clusters en el entrenamiento $\mathcal{S}_{v_0=0.75}$ es mucho mayor que el de $\mathcal{S}_{v_0=0.5}$ y el predictor de *SVM* y *RF* entrenado con $\mathcal{S}_{v_0=0.75}$ asocia el cluster 2 con el cluster 1 porque la lejanía con el cluster 2 es excesiva.

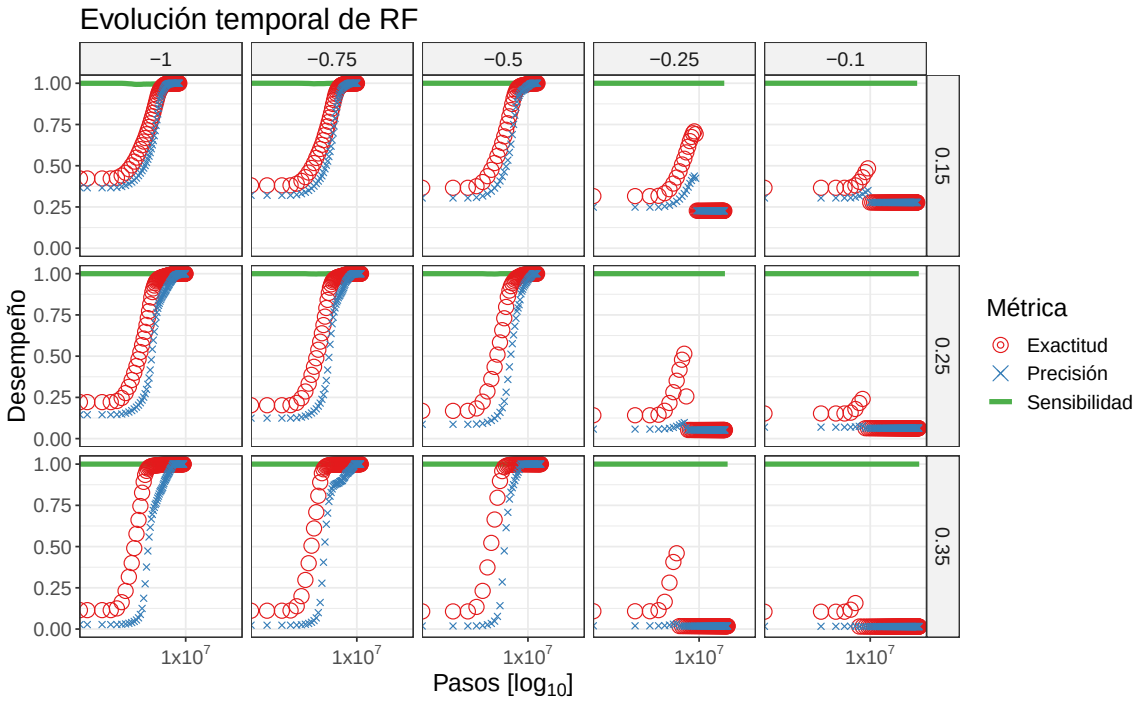

Figura 4.20.: Evolución de las métricas del predictor de *RF* en función de pasos temporales para los 15 casos evaluados de la extensión local. Los casos están graficados por las $\vec{v_0}$ (columnas) y los ϕ (filas). La evolución de los 15 casos evaluados usando *SVM* son semejantes.

Nuevamente *OCSVM* clasifica bien todos los casos, generalizando su predictor más allá de los casos con los cuales entrenó. Se muestran el ahorro y la ganancia temporal para *OCSVM* en la siguiente tabla:

$T_{n_{peq}}$	T_e	% Ex.	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	1d 2H 33M	90 %	3d 17H 33M	12d 5H 45M	76.6 %
		95 %	3d 23H 15M	12d 0H 3M	75.1 %
		99 %	-	-	-

Tabla 4.7: Tabla de costos y ahorros temporales de *OCSVM* entrenado con $\mathcal{S}_{v_0=0.75}$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_{v_0=0.75}$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}$−T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

Los casos en los cuales *OCSVM* no llega al 99 % de exactitud son ($0.5\frac{m}{s}$, 0.15), ($0.75\frac{m}{s}$, 0.15) y ($1\frac{m}{s}$, 0.15), nuevamente alcanzando solamente un 97 % de exactitud.

Resultados de $\mathcal{S}_{v_0=1}$

Por la forma del entrenamiento $\mathcal{S}_{v_0=1}$ y el comportamiento que se tuvo en las otras pruebas por v_0, se esperaba que los algoritmos entrenados con él tuviesen el peor desempeño, puesto que sus centroides están muy separados y sus elementos en ambos clusters no están claramente delimitados, por lo que tampoco puede distinguir muy bien el cluster 1 (Fig. 4.16). Como se muestra en la Fig. 4.21, la evolución de las métricas de *RF* (y de *SVM*, pues son similares) es incorrecta para más de la mitad de los casos. Las velocidades finales en z del cluster 2 de los casos mal clasificados son menores en módulo a $0.049\frac{m}{s}$. Ya el caso ($0.5\frac{m}{s}$, 0.15) posee una velocidad en z final del cluster 2 de magnitud $0.064\frac{m}{s}$, y el algoritmo logra reconocerlo como cluster 2 (Fig. 3.5).

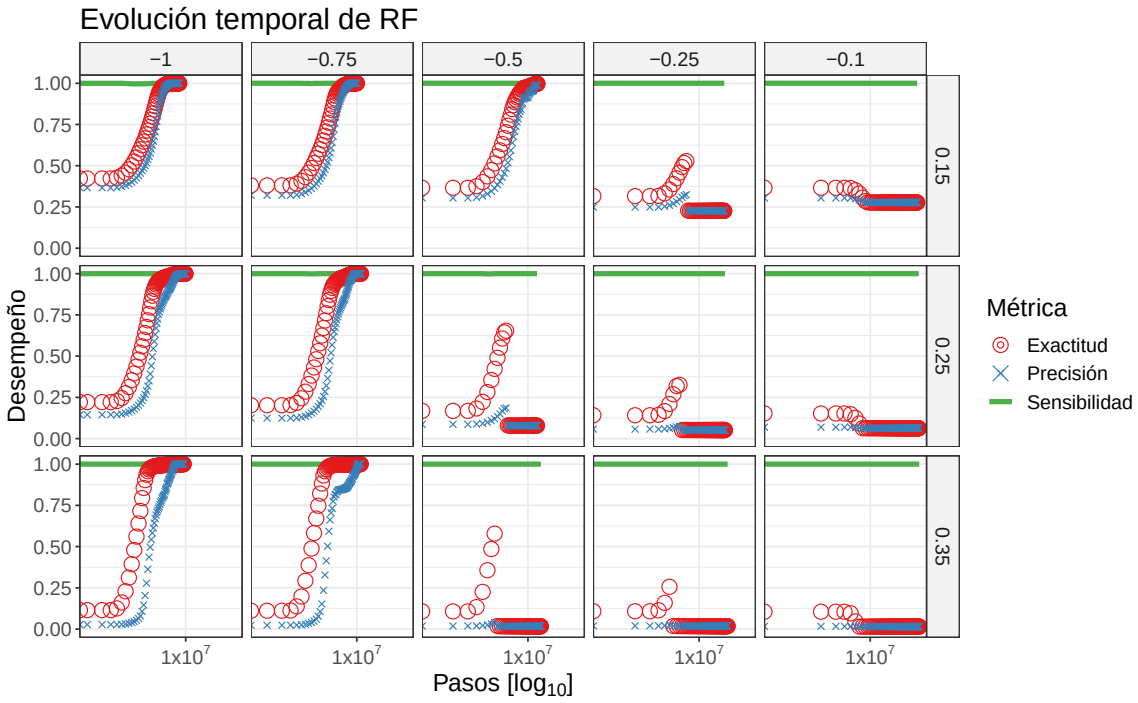

Figura 4.21.: Evolución de las métricas del predictor de *RF* en función de pasos temporales para los 15 casos evaluados de la extensión local. Los casos están graficados por las $\vec{v_0}$ (columnas) y los ϕ (filas). La evolución de los 15 casos evaluados usando *SVM* son semejantes.

A pesar del mal desempeño de *RF* y *SVM*, *OCSVM* se desempeña bien nuevamente en los 15 casos. No solamente esto, también esta es la única prueba en la cual todas las predicciones alcanzan el 99 % de exactitud. Esto es bastante curioso, pues se esperaría que, como el algoritmo aprende solamente los puntos del cluster 1, todos los puntos del resto de los casos cuyos cluster 2 caigan en el mismo intervalo de v_z del cluster 1 de $\mathcal{S}_{v_0=1}$ (Fig. 4.16), sean mal clasificados, especialmente los casos $(0.1\frac{m}{s}, \phi)$. Sin embargo, la concentración de partículas en el cluster 1 del entrenamiento $\mathcal{S}_{v_0=1}$ que están en la zona de superposición con la velocidad final promedio de los cluster 2 de los casos $(0.1\frac{m}{s}, \phi)$, es en realidad bastante menos densa de lo que parece en la figura. Para visualizar esto, en la figura 4.22 se muestra el cluster 1 del entrenamiento $\mathcal{S}_{v_0=1}$ junto a las velocidades finales de los tres cluster 2 correspondientes a los casos $(0.1\frac{m}{s}, \phi)$. Puede apreciarse en la figura que las partículas del cluster 1 que

se superponen a las del cluster 2 son relativamente pocas, y por ende la clasificación de *OCSVM* alcanza el 99 % de exactitud.

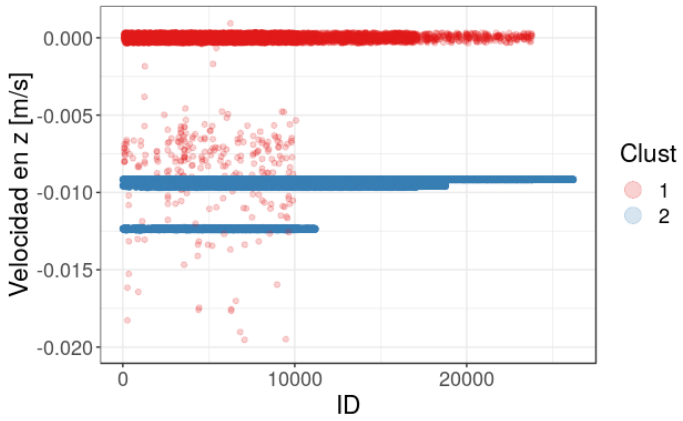

Figura 4.22.: Se muestra la superposición de las partículas clasificadas como cluster 1 en el entrenamiento $\mathcal{S}_{v_0=1}$ y aquellas clasificadas como cluster 2 en los tres casos $(0.1\frac{m}{s}, \phi)$.

Se muestra el porcentaje de ganancia de *OCSVM* en la tabla 4.8:

$T_{n_{peq}}$	T_e	% Ex.	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	21H 3M	90 %	3d 13H 15M	12d 10H 3M	77.8 %
		95 %	3d 19H 1M	12d 4H 18M	76.3 %
		99 %	4d 8H 55M	11d 14H 23M	72.6 %

Tabla 4.8: Tabla de costos y ahorros temporales de *OCSVM* entrenado con $\mathcal{S}_{v_0=1}$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_{v_0=1}$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

Notar que esta es la prueba que más porcentaje de ganancia tiene, y esto es debido a que las simulaciones con velocidad $v_0 = 1\frac{m}{s}$ son las que menos tiempo demoran en simularse hasta el final.

Conclusiones para pruebas por v_0

El entrenamiento que permitió un buen desempeño de los tres algoritmos en los 15 casos fue el de $\mathcal{S}_{v_0=0.1}$. Sin embargo, comparando todos los tiempos de entrenamiento T_e para cada una de las cinco pruebas por v_0, este es el que más demora en simularse, y por lo tanto el que menos porcentaje de ganancia presenta. Si observamos los resultados de ganancia temporal al utilizar *OCSVM*, este va en aumento para la misma exactitud cuanto mayor sea el v_0 del entrenamiento, como se muestra en la Fig. 4.23. No es inesperado que el mayor porcentaje de ahorro temporal de simulación lo tenga el entrenamiento $\mathcal{S}_{v_0=1}$, pues las simulaciones que se necesitan simular hasta el final para poder conformar el entrenamiento son las que menos demoran en tiempo real, y las simulaciones sobre las que se predice el desenlace de la colisión poseen el mayor tiempo reloj de simulado, y este entrenamiento evita tener que ejecutarlas hasta el fin. En cambio, el menor ahorro se produce para el entrenamiento $\mathcal{S}_{v_0=0.1}$, pues el tiempo de entrenamiento considera el tiempo reloj de simulado hasta el final de todas las simulaciones más largas.

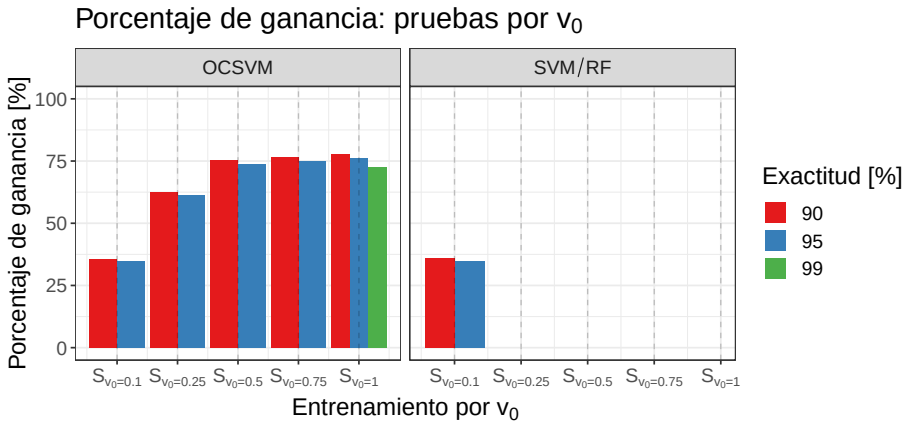

Figura 4.23.: Porcentaje de ganancia temporal para los predictores de los algoritmos entrenados mediante entrenamientos de la forma $\mathcal{S}_{v_0=*}$ que logran generalizar las predicciones a toda la extensión local.

El entrenamiento que da cuenta del resto de los casos de manera satisfactoria para los tres algoritmos es el que posee los centroides de sus clusters más cercanos,

$\mathcal{S}_{v_0=0.1}$. El éxito de las predicciones de *SVM* y *RF* con este entrenamiento se debe a dos fenómenos: el primero es que las velocidades finales en z de los cluster 2 de los casos no involucrados en el entrenamiento son mayores en magnitud a las de los casos entrenados, y el segundo es que no hay superposiciones significativas de clusters distintos de los otros casos.

Para *OCSVM*, como tuvo buen desempeño con todos los entrenamientos, este no fue el de mejor resultados.

4.3.3 Pruebas combinadas

Del resultado de los entrenamientos por ϕ y v_0, se estima que un entrenamiento $\mathcal{S}_1$ con todos los casos de $v_0 = 0.1\frac{m}{s}$ y todos los casos de $\phi = 0.35$ (es decir, $\mathcal{S}_1 = \mathcal{S}_{v_0=0.1} \cup \mathcal{S}_{\phi=0.35}$), debería lograr un buen desempeño de los algoritmos. Se muestra este conjunto de entrenamiento en la Fig. 4.24.

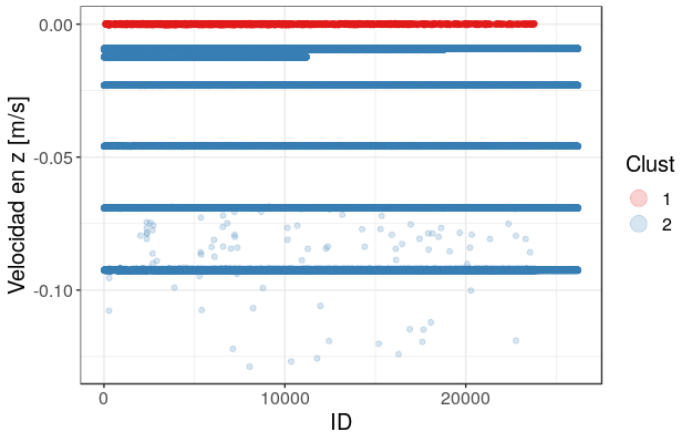

Figura 4.24.: Se muestra la variable predictora $\vec{v_z}$ en función de el ID para el conjunto de entrenamiento $\mathcal{S}_1$: partículas correspondientes a las configuraciones finales de las simulaciones de los entrenamientos $\mathcal{S}_1 = \mathcal{S}_{v_0=0.1} \cup \mathcal{S}_{\phi=0.35}$ coloreadas por pertenencia a cluster. Se distinguen seis de los siete cluster 2 (grupos horizontales azules) correspondientes a las simulaciones integrantes de este entrenamiento, pues uno está superpuesto sobre el primer grupo.

Como puede observarse, los datos están linealmente separados y los clusters están bien delimitados, y por ende se entrenó *SVM* con núcleo lineal. Los resultados de la evolución temporal de las métricas se muestran en la Fig. 4.25.

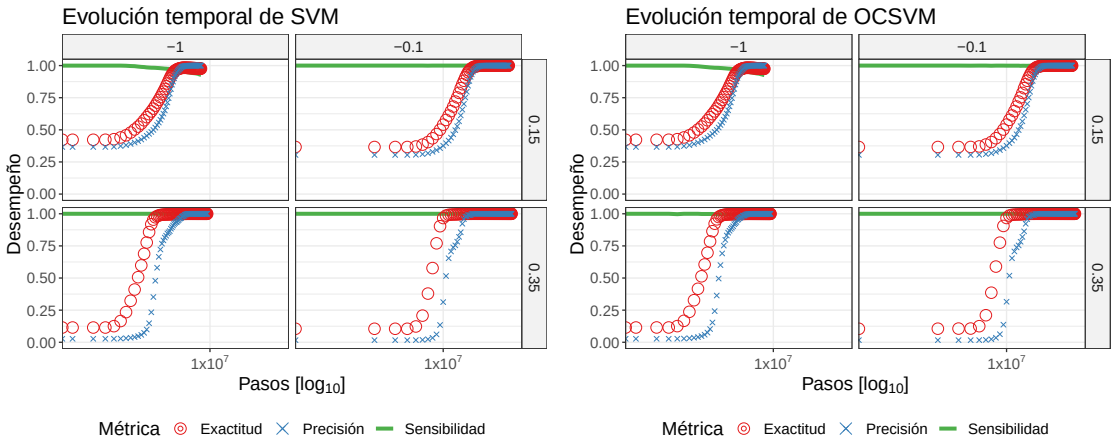

Figura 4.25.: Evolución de las métricas para los algoritmos *SVM* y *OCSVM* en función de los pasos temporales. Nuevamente observamos el comportamiento característico de la sensibilidad y la precisión. Los casos están graficados por las $\vec{v_0}$ (columnas) y los ϕ (filas). Las métricas de *RF* evolucionan de forma semejante a *SVM*.

El desempeño de los algoritmos y la evolución de sus métricas es semejante a lo observado en pruebas anteriores exitosas.

Los tres algoritmos logran generalizar mediante sus predictores las clasificaciones del resto de los casos, y el ahorro temporal de cada algoritmo se detalla a continuación:

$T_{n_{peq}}$	T_e	% Ex.	Algoritmo	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	11d 16H 20M	90 %	*SVM*	14d 7H 30M	1d 15H 49M	10.4 %
			RF	14d 7H 30M	1d 15H 49M	10.4 %
			OCSVM	14d 7H 30M	1d 15H 49M	10.4 %
		95 %	*SVM*	14d 13H 13M	1d 10H 6M	8.9 %
			RF	14d 13H 13M	1d 10H 6M	8.9 %
			OCSVM	14d 13H 53M	1d 9H 25M	8.7 %
		99 %	*SVM*	-	-	-
			RF	-	-	-
			OCSVM	-	-	-

Tabla 4.9: Tabla de costos y ahorros temporales de los tres algoritmos entrenado con $\mathcal{S}_1$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_1$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

El único caso que no alcanza al 99 % de exactitud, para los tres algoritmos, es ($1\frac{m}{s}$, 0.15) (logra un valor de 98 %). Esto se debe a un pequeño sobreajuste en los algoritmos, presente en todas las pruebas anteriores.

Como puede observarse, el ahorro temporal no es nada satisfactorio, de las simulaciones de entrenamiento los casos del entrenamiento S_1 son los que más tiempo tardan en ejecutar hasta T_f, por lo tanto, predecir el resto de las simulaciones mediante este entrenamiento no es muy óptimo.

Se fue al extremo de intentar la intersección entre los casos de $\mathcal{S}_{v_0=0.1}$ y los casos de $\mathcal{S}_{\phi=0.35}$, es decir, solamente entrenar el caso de ($0.1\frac{m}{s}$, 0.35). Este entrenamiento se denominará S_2 y se esquematiza en la Fig. 4.26:

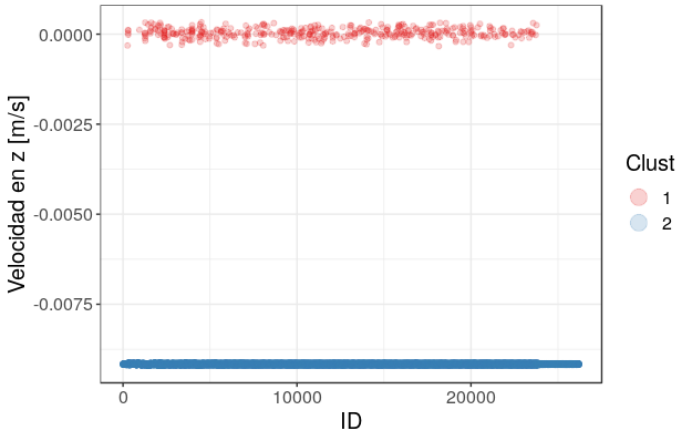

Figura 4.26.: Se muestra la variable predictora $\vec{v_z}$ en función de el ID para el conjunto de entrenamiento $\mathcal{S}_2$: partículas correspondientes a las configuraciones finales de la simulación $(0.1\frac{m}{s}, 0.35)$ coloreadas por pertenecia a cluster.

Sorprendentemente, los tres algoritmos evolucionan de manera correcta, a pesar de la poca densidad de partículas que se tienen en cada cluster (Fig. 4.26). Los resultados de la evolución temporal de las métricas se muestran en la Fig. 4.27.

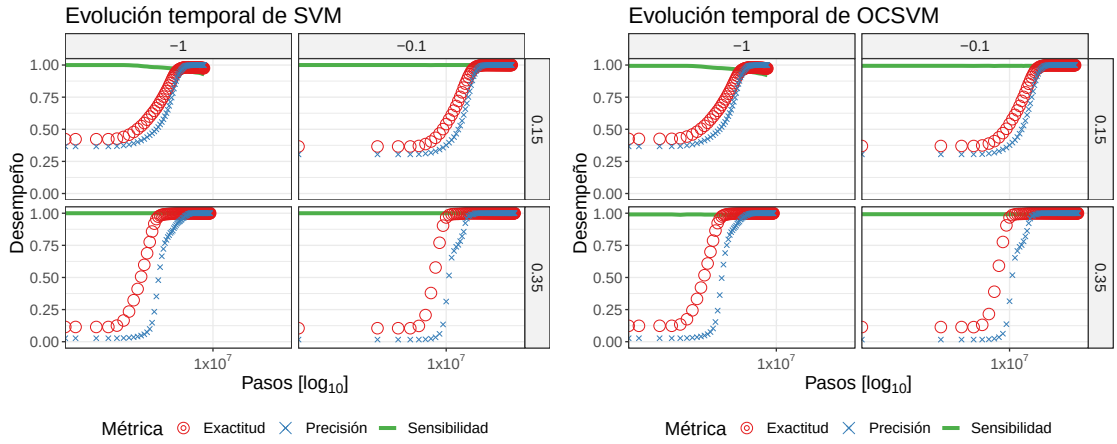

Figura 4.27.: Evolución de las métricas para los algoritmos *SVM* y *OCSVM* en función de los pasos temporales. Nuevamente observamos el comportamiento característico de la sensibilidad y la precisión. Los casos están graficados por las $\vec{v_0}$ (columnas) y los ϕ (filas). Las métricas de *RF* evolucionan de forma semejante a *SVM*.

Es evidente que con estos algoritmos se demorará menos tiempo en tener la configuración final de las 15 simulaciones que con el entrenamiento $\mathcal{S}_1$, pues solo se necesita

simular hasta el final el único caso del entrenamiento. El ahorro temporal se muestra en la siguiente tabla:

$T_{n_{peq}}$	T_e	% Ex.	Algoritmo	T	$T_{n_{peq}} - T$	G_t
15d 23H 19M	2d 15H 1M	90 %	*SVM*	3d 17H 34M	12d 5H 45M	76.6 %
			RF	3d 17H 34M	12d 5H 45M	76.6 %
			OCSVM	3d 18H 4M	12d 5H 14M	76.5 %
		95 %	*SVM*	3d 20H 18M	12d 3H 0M	75.9 %
			RF	3d 20H 18M	12d 3H 0M	75.9 %
			OCSVM	3d 21H 2M	12d 2H 17M	75.7 %
		99 %	*SVM*	-	-	-
			RF	-	-	-
			OCSVM	-	-	-

Tabla 4.10: Tabla de costos y ahorros temporales de los tres algoritmos entrenado con $\mathcal{S}_2$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_2$

, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{peq}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

El único caso que no llega al 99 % de exactitud para ninguno de los 3 algoritmos es $(1\frac{m}{s}, 0.15)$, igual que con el entrenamiento $\mathcal{S}_2$. Esto es debido al pequeño sobreajuste del que se ha hablado anteriormente, la curvatura que se produce en la exactitud producto del aumento de la tasa de FN impide que el predictor supere el 98 % de exactitud en este caso.

Conclusiones de las pruebas combinadas

Las ganancias temporales en cada una de las dos pruebas presentadas en esta sección son casi iguales para los tres algoritmos. Evidentemente conviene usar el entrenamiento S_2 en vez del S_1, pues ahorra $\sim 60\,\%$ más tiempo que este último.

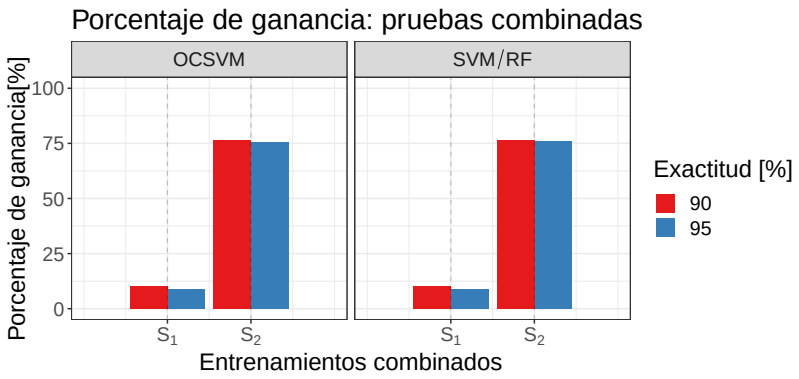

Figura 4.28.: Porcentaje de ganancias temporales para $\mathcal{S}_1$ y $\mathcal{S}_2$. Los porcentajes son idénticos en el caso de *SVM* y *RF*, y estos muy parecidos con *OCSVM*.

Será casi indistinto que el analista corra las simulaciones hasta un 90 % o un 95 % de exactitud, pero se recomienda un 95 % para tener mayor confianza en los datos (mayor precisión).

4.4 Extensión distante

El objetivo de la extensión distante es encontrar un entrenamiento formado de las configuraciones finales de alguna de las simulaciones de entrenamiento de tamaño pequeño, y a partir de él lograr predecir el desenlace de simulaciones de tamaño mediano y grande. En total se probaron cinco entrenamientos, de los cuales cuatro se tomaron de la extensión local ($\mathcal{S}_{\phi=0.35}$, $\mathcal{S}_{v_0=0.1}$, $\mathcal{S}_1$, $\mathcal{S}_2$) y uno no utilizado anteriormente para poder evaluar los casos de tamaño grande. Los entrenamientos se esquematizaron en la Fig. 4.1.

4.4.1 Extensión distante: tamaño mediano

Se disponen de 15 simulaciones de tamaño mediano, especificadas en la Sección 3.4.3. Las simulaciones tienen los mismos casos que las de tamaño pequeño, solamente cambia el parámetro general tamaño de proyectil/blanco (aumentando el número de partículas del blanco). Se denominará $T_{n_{med}}$ (calculado mediante Ec. 3.9) al tiempo

total real de simulación requerido para obtener la configuración final de las 15 simulaciones de tamaño mediano. Se tiene que $T_{n_{med}} = 32\text{d } 14\text{H } 55\text{M}$, lo cual es un poco más del doble del tiempo para obtener las simulaciones pequeñas.

Los entrenamientos elegidos para realizar las pruebas sobre este tamaño son cuatro: $\mathcal{S}_{\phi=0.35}$, $\mathcal{S}_{v_0=0.1}$, $\mathcal{S}_1$, $\mathcal{S}_2$.

Resultados de $\mathcal{S}_{\phi=0.35}$, $\mathcal{S}_{v_0=0.1}$, $\mathcal{S}_1$ y $\mathcal{S}_2$

Para todos los entrenamientos evaluados en las simulaciones de tamaño mediano, ni *RF* ni *SVM* tuvieron un buen desempeño. Si se sigue el comportamiento de estos algoritmos en estas pruebas y sus desempeños sobre simulaciones de tamaño mediano, se puede inferir que la raíz del problema está en la variable predictora v_z. Estos algoritmos deben aprender a reconocer dos clusters: el cluster 1 siempre tendrá una v_z promedio cercana a cero, pero como el cluster 2 varía según los tres parámetros generales tratados a lo largo del trabajo (tamaño proyectil/blanco, v_0 y ϕ), las simulaciones evaluadas que poseen un cluster 2 con v_z final considerablemente mayor al de su entrenamiento tienden a dar malos resultados. Si se observa la Fig. 4.29, se puede comparar la diferencia entre las velocidades finales en z de los cluster 2 entre las simulaciones de entrenamiento de tamaño pequeño y las simulaciones evaluadas de tamaño mediano.

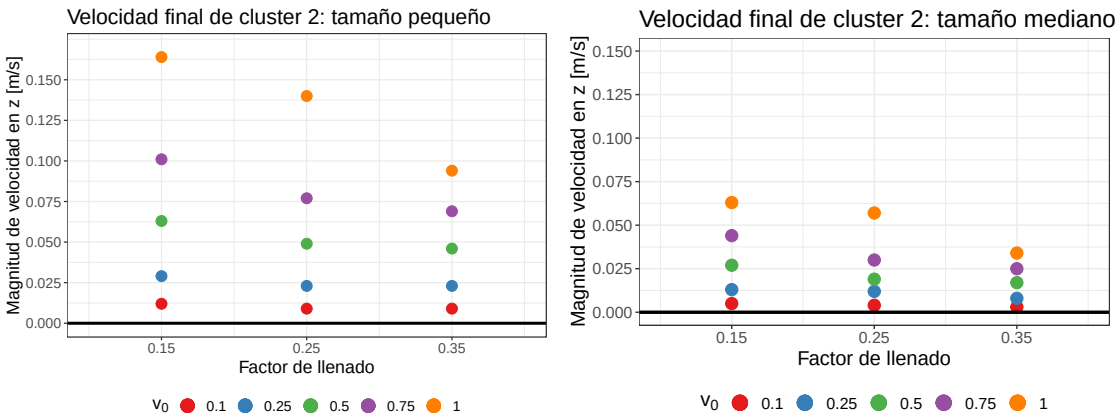

Figura 4.29.: Velocidad final en z (variable predictora) de los cluster 2 de las simulaciones de entrenamiento (tamaño pequeño) y las simulaciones evaluadas (tamaño mediano.)

Tal como puede apreciarse en la Fig. 4.29, no se espera que ninguno de los entrenamientos logren dar cuenta de las simulaciones con v_0 muy pequeñas, pues la cercanía de las v_z de los cluster 2 para estos casos evaluados con el cluster 1 de los entrenamientos es muy grande.

En la siguiente figura se muestra un ejemplo representativo de las falencias de ambos algoritmos. Todas las pruebas dieron resultados similares, para los mismos casos.

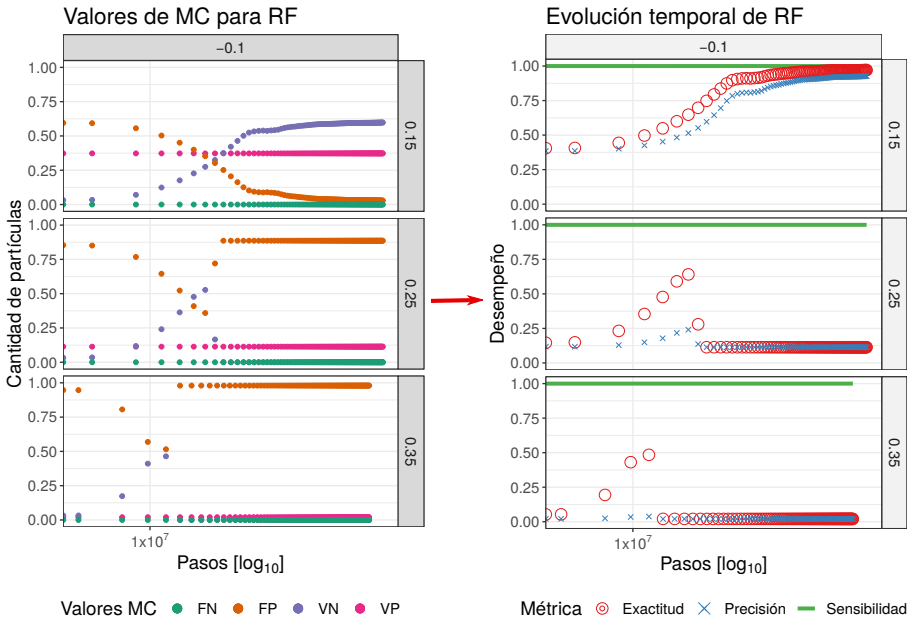

Figura 4.30.: Evolución incorrecta de las métricas para *RF* (*SVM* se comporta de forma similar). Valores de la MC en función de los pasos temporales (izq.) para casos de evoluciones no correctas de *RF* (der.). *SVM* tiene problemas con los mismos casos, con características similares. Los gráficos están divididos por $\vec{v_0}$ (columnas) y ϕ (filas). Notar que los pasos temporales están en escala logarítmica.

Aunque el caso $(0.1\frac{m}{s}, 0.15)$ llega a tener una exactitud de mayor al 95 %, la curva no es la sigmoidea suave que se busca. La tasa de VP va disminuyendo en valor con el aumento del ϕ en los distintos casos mostrados en la figura debido a que a mayor ϕ menos partículas quedan en el cluster 1, por lo tanto, cuando en el primer paso del caso $(1\frac{m}{s}, 0.15)$ el algoritmo clasifica a todas las partículas del blanco como parte del cluster 1, acierta en más partículas que en el paso cero de los otros dos casos.

Los casos de velocidades $v_0 = 0.1\frac{m}{s}$ no están bien representadas por estos entrenamientos, tal como se dijo, los clusters finales de este tamaño son muy distintos a los del tamaño pequeño como para que *SVM* o *RF* puedan clasificarlos correctamente.

OCSVM, sin embargo, vuelve a tener un buen desempeño en general, aunque vuelve a tener una pequeña tendencia en su exactitud curvada hacia abajo, producto de un sobreajuste, que nuevamente influye en la exactitud.

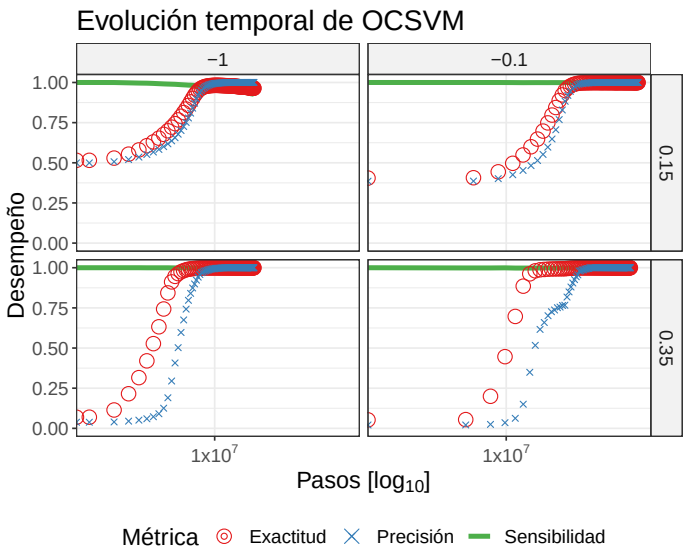

Figura 4.31.: Evolución de las métricas para los cuatro casos extremos de *OCSVM*. En las columnas se muestran dos $\vec{v_0}$ y en las filas, dos valores de ϕ. La evolución del resto de los casos es semejante.

Se muestra la ganancia de *OCSVM* en las siguientes tablas. Debe notarse que el tiempo de entrenamiento T_e es el de las pruebas de tamaño pequeño. Las tablas 4.11, 4.12, 4.13, 4.14 especifican los siguientes valores; T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_*$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{med}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

$T_{n_{med}}$	T_e	% Ex.	T	$T_{n_{med}} - T$	G_t
32d 14H 55M	5d 13H 15M	90 %	9d 10H 7M	23d 4H 48M	71.1 %
		95 %	9d 19H 17M	22d 19H 37M	69.9 %
		99 %	-	-	-

Tabla 4.11: Ahorro temporal de *OCSVM* con el entrenamiento $\mathcal{S}_{\phi=0.35}$.

$T_{n_{med}}$	T_e	% Ex.	T	$T_{n_{med}} - T$	G_t
32d 14H 55M	8d 18H 7M	90 %	12d 14H 59M	19d 23H 56M	61.3 %
		95 %	13d 0H 43M	19d 14H 11M	60.1 %
		99 %	-	-	-

Tabla 4.12: Ahorro temporal de *OCSVM* con el entrenamiento $\mathcal{S}_{v_0=0.1}$.

$T_{n_{med}}$	T_e	% Ex.	T	$T_{n_{med}} - T$	G_t
32d 14H 55M	11d 16H 20M	90 %	15d 13H 12M	17d 1H 43M	52.3 %
		95 %	15d 22H 22M	16d 16H 32M	51.2 %
		99 %	-	-	-

Tabla 4.13: Ahorro temporal de *OCSVM* con el entrenamiento $\mathcal{S}_1$.

$T_{n_{med}}$	T_e	% Ex.	T	$T_{n_{med}} - T$	G_t
32d 14H 55M	2d 15H 1M	90 %	6d 11H 53M	26d 3H 2M	80.1 %
		95 %	6d 21H 47M	25d 17H 8M	78.8 %
		99 %	-	-	-

Tabla 4.14: Ahorro temporal de *OCSVM* con el entrenamiento $\mathcal{S}_2$.

OCSVM apenas supera el 95 % de exactitud en los entrenamientos $\mathcal{S}_{\phi=0.35}$, $\mathcal{S}_{v_0=0.1}$ y $\mathcal{S}_1$ en los casos de tamaño mediano ($0.75\frac{m}{s}$,0.15) y ($1\frac{m}{s}$, 0.15) debido al sobreajuste de los predictores. Para $\mathcal{S}_2$ también alcanza hasta este valor en estos dos casos y en ($1\frac{m}{s}$,0.25).

Conclusiones de pruebas sobre tamaño mediano

A pesar de que los algoritmos entrenados con $\mathcal{S}_{\phi=0.35}$, $\mathcal{S}_{v_0=0.1}$, $\mathcal{S}_1$ y $\mathcal{S}_2$ dan cuenta de la distribución de los 15 casos en la extensión local, para la evaluación sobre el

tamaño mediano *RF* y *SVM* fallan en la velocidad $v_0 = 0.1\frac{m}{s}$ debido a la diferencia de cercanía entre los centroides del cluster 1 y cluster 2 en el entrenamiento y las simulaciones evaluadas. La variación de la velocidad final en z de los cluster 2 al cambiar el tamaño de banco/proyectil en las simulaciones dificulta la distinción correcta de los dos clusters para *RF* y *SVM*.

OCSVM, tal como se esperaba, funciona correctamente exceptuando un sobreajuste en el caso $(1\frac{m}{s}, 0.15)$, el mismo que se observa en la extensión local. El entrenamiento sobre *OCSVM* que más tiempo ahorra al dar cuenta de la configuración del resto de las simulaciones es $\mathcal{S}_2$.

El porcentaje de ganancia logrado por *OCSVM* de estas pruebas se presenta en la siguiente tabla:

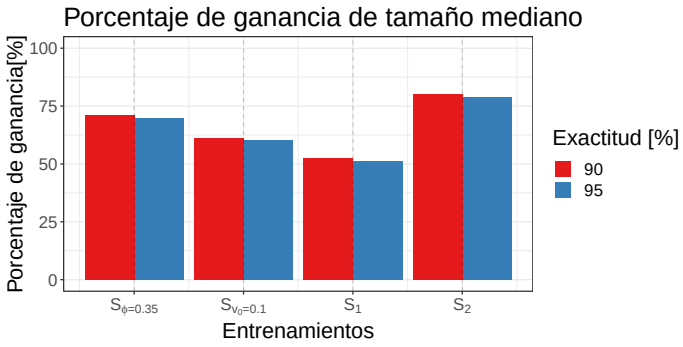

Figura 4.32.: Porcentaje de ganancia temporal para los entrenamientos de la forma $\mathcal{S}_*$, para tamaño mediano logrados por *OCSVM*.

4.4.2 Extensión distante: tamaño grande

Del tamaño grande de proyectil/blanco sólo se testearon cinco casos: $(0.1\frac{m}{s}, \phi = 0.15)$, $(0.25\frac{m}{s}, \phi = 0.15)$, $(0.5\frac{m}{s}, \phi = 0.15)$, $(0.75\frac{m}{s}, \phi = 0.15)$ y $(1\frac{m}{s}, \phi = 0.15)$ (ver Sección 3.4.3), debido al extenso tiempo real que lleva ejecutar este tamaño de simulaciones. El tiempo real total que demorado en ejecutar estas cinco simulaciones de tamaño grande es $T_{n_{gr}}$ = 14d 1H 16M. De haber corrido los 15 casos para este tamaño, este habría superado el tiempo T_n de los otros tamaños.

Se utilizaron dos entrenamientos, $\mathcal{S}_2$ y uno no usado anteriormente; $\mathcal{S}_3$, el cual posee los datos etiquetados de la configuración final del caso de tamaño pequeño $(0.1\frac{m}{s}, 0.15)$. Se eligieron estos entrenamientos por la cantidad de casos en los cuales se deseaba evaluar los predictores. Si, por ejemplo, se utilizaba $\mathcal{S}_1$ no se habría ahorrado tiempo significativo, pues T_e es de aproximadamente 11 días y medio, y si se le agregase el tiempo empleado predecir cada archivo de salida de los casos de evaluación, el tiempo resultante sería muy cercano a $T_{n_{gr}} \sim 14$ días y por ende, el porcentaje de ganancia sería casi nulo.

Resultados de $\mathcal{S}_2$ y $\mathcal{S}_3$

Ambas pruebas dieron resultados muy similares para los tres algoritmos. *RF* y *SVM* tienen un buen desempeño en todos los casos menos en $(0.1\frac{m}{s}, 0.15)$. El mal desempeño de estos dos algoritmos en este caso es debido a que en este tamaño de simulaciones la velocidad final en z de los cluster 2 de las simulaciones evaluadas son aún más pequeñas que los de tamaño mediano, y por ende los algoritmos no pueden distinguir este caso. Esta hipótesis la refuerza el hecho de que *OCSVM* clasifica bien todos los casos evaluados nuevamente (Fig. 4.33).

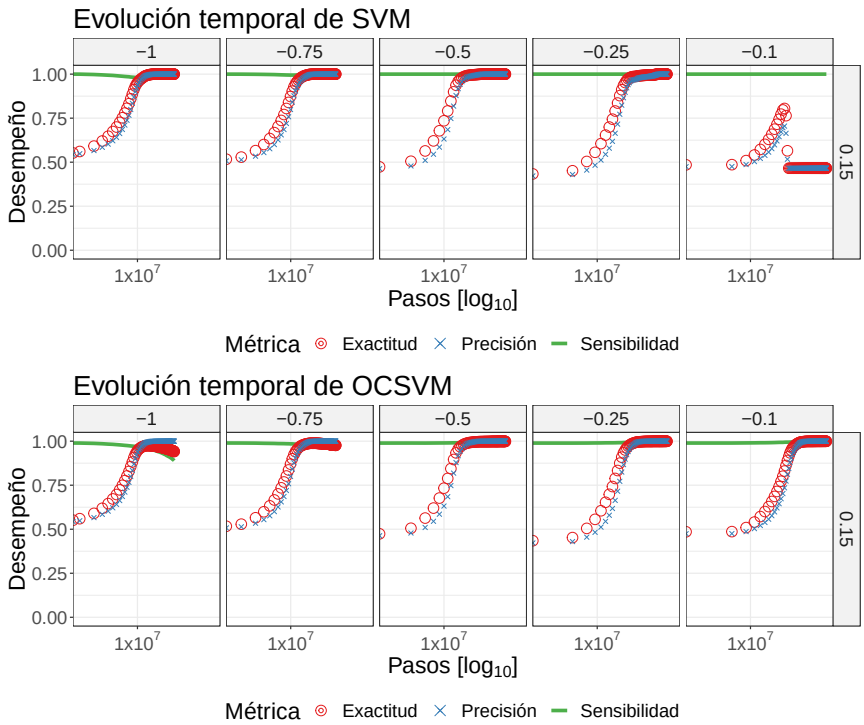

Figura 4.33.: Evolución para *SVM* y *OCSVM*. Las dos pruebas tienen como resultado formas muy parecidas y la evolución de las métricas en *RF* es casi idéntica a *SVM*.

OCSVM, a pesar de superar el 95 % de exactitud en todos los casos, posee una curvatura más marcada de esta métrica para el caso $(1\frac{m}{s}, 0.15)$ que para el caso $(0.75\frac{m}{s}, 0.15)$. En el primero, la sensibilidad disminuye hasta un valor de 89 % en el último paso debido a la disminución de VP, por lo cual también la exactitud posee un valor de 94 %. Para el caso $(0.75\frac{m}{s}, 0.15)$ la sensibilidad en el último paso es de 95 % y la exactitud de 97 %. Los ahorros temporales de *OCSVM* se detalla en la siguiente tabla:

$T_{n_{gr}}$	T_e	% Ex.	T	$T_{n_{gr}} - T$	G_t
14d 1H 16M	2d 15H 1M	90 %	5d 2H 12M	8d 23H 4M	63.8 %
		95 %	5d 13H 30M	8d 11H 45M	60.4 %
		99 %	-	-	-

Tabla 4.15: Ahorro temporal de los tres algoritmos con el entrenamiento $\mathcal{S}_2$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_2$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{gr}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

Los casos de tamaño grande evaluados con el entrenamiento $\mathcal{S}_2$ en los cuales *OCSVM* no se alcanzó en 99 % de exactitud fueron ($0.75\frac{m}{s}$,0.15) y ($1\frac{m}{s}$, 0.15).

$T_{n_{gr}}$	T_e	% Ex.	T	$T_n - T$	G_t
14d 1H 16M	2d 13H 21M	90 %	5d 0H 32M	9d 0H 44M	64.3 %
		95 %	5d 11H 50M	8d 13H 25M	60.9 %
		99 %	-	-	-

Tabla 4.16: Ahorro temporal de los tres algoritmos con el entrenamiento $\mathcal{S}_3$. T_e: tiempo real de entrenamiento de las simulaciones participantes en $\mathcal{S}_3$, % Ex.: porcentaje de exactitud, T: tiempo empleado en predecir el desenlace de las 15 simulaciones, $T_{n_{gr}}-$T: tiempo real ahorrado de simulación, G_t: porcentaje de ganancia.

El único caso de tamaño grande evaluado con este entrenamiento en el cual *OCSVM* no alcanzó en 99 % de exactitud fue ($1\frac{m}{s}$, 0.15).

Conclusiones para pruebas sobre tamaño grande

Estas pruebas muestran cómo *OCSVM* funcionaría correctamente aún en este tamaño de simulaciones, aunque se necesita encontrar una manera de corregir el sobreajuste que se produce en las velocidades mayores.

El porcentaje de ganancia temporal para *OCSVM* se muestra en la siguiente figura:

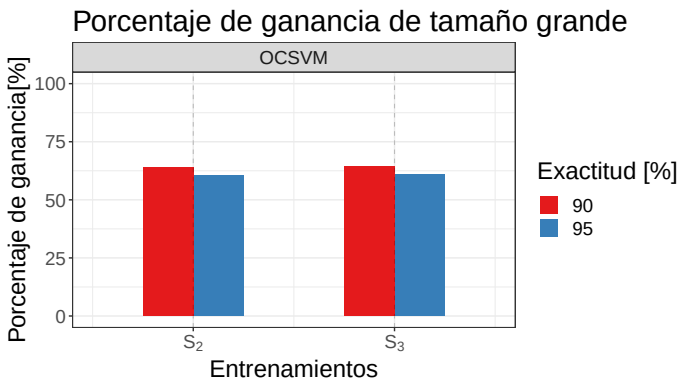

Figura 4.34.: Porcentaje de ganancia temporal para los entrenamientos de la forma $\mathcal{S}_*$, para tamaño grande.

Los porcentajes de ganancia son muy parecidos, y esta es una de las razones por las cuales no se probó el entrenamiento $\mathcal{S}_3$ en la extensión local.

De poseer las 15 simulaciones de tamaño grande (los restantes 10 casos (v_0, 0.25) y (v_0, 0.35)), el tiempo ahorrado por *OCSVM* sería mucho mayor que para las simulaciones de tamaño mediano, pues el tiempo reloj que se demora en ejecutar todas estas simulaciones grandes es de aproximadamente dos meses.

4.5 Tiempo de etiquetado y entrenamiento

Por supuesto, etiquetar los casos que son parte del entrenamiento, entrenar cada algoritmo y evaluarlos en 100 archivos de salida de cada simulación de evaluación también tiene su costo temporal. Ninguna de estas pruebas tendría sentido si se le ofrece al analista una forma de ahorrar 12 días de simulación, pero aclarándole que el etiquetado de datos y el entrenamiento de los algoritmos demora 12 días más en realizarse.

Etiquetar los datos de entrenamiento con *k-means* no supone una adición temporal significativa, pues cada etiquetado demora alrededor de diez segundos en implementarse en cada caso. La ventaja de todas las pruebas realizadas en las secciones anteriores es que el tiempo de entrenamiento de cada algoritmo mediante $\mathcal{S}$ es en promedio menor a 1 minuto, salvo algunas pocas excepciones. Por ejemplo, *SVM* tiene

un tiempo de entrenamiento para el caso $\mathcal{S}_{\phi=0.25}$ de 9 minutos, porque aplica el truco de núcleo radial (una transformación matemática de los datos) sobre una cantidad considerable de partículas.

En general, comparando el tiempo que demora entrenar cada algoritmo en todos los $\mathcal{S}$, el menor tiempo de entrenamiento lo posee *OCSVM*, puesto que sólo entrena con los datos etiquetados como cluster 1, y ninguno de los 11 entrenamientos de este algoritmo supera los 0.8 segundos. Realizando una comparación de las pruebas con evoluciones correctas para los tres algoritmos ($\mathcal{S}_{\phi=0.35}$, $\mathcal{S}_{v_0=0.1}$, $\mathcal{S}_1$, $\mathcal{S}_2$), se observa que en promedio los algoritmos demoran mayor cantidad de tiempo entrenando mediante $\mathcal{S}_{\phi=0.35}$ y $\mathcal{S}_{v_0=0.1}$.

Una vez que entrena los algoritmos, el programa "lee" cada archivo de salida, es decir, abre el archivo, lo convierte en un formato especial para poder ser interpretado en R, predice las categorías de las partículas (cluster 1 o 2) para cada archivo mediante el predictor h resultante de la etapa de entrenamiento y evalúa la Matriz de Confusión. Para las simulaciones de tamaño pequeño, el programa no tardó más de dos minutos en promedio para realizar este proceso de lectura, y para los tamaños mediano y grande no sobrepasó los dos minutos y medio. Dentro de este tiempo de lectura el tiempo de predicción y evaluación es, para casi todos los algoritmos y todos los tamaños, menor al segundo. Sólo hay una excepción no mucho mayor de 1.5 segundos. En promedio, el algoritmo que menos tiempo tarda en predecir y evaluar los resultados de su predictor h es *RF*, seguido por *OCSVM* y por último *SVM*.

En la siguiente sección se muestra el ahorro temporal de un sistema de AA completo para el algoritmo *OCSVM* considerando también los tiempos aquí descritos.

4.6 Elección de *OCSVM*

El algoritmo *OCSVM* se desempeña correctamente con todos los entrenamientos probados. Aún si su tolerancia de error no le permite llegar al 100 % de exactitud, siempre supera el 95 %. Si tomamos sólo este algoritmo, el entrenamiento con mayor

ganancia temporal para la extensión local es $\mathcal{S}_{v_0=1}$ (considerando sólo exactitudes del 90 % y 95 %), y el de menor ganancia es $\mathcal{S}_1$ como muestra la Fig. 4.35.

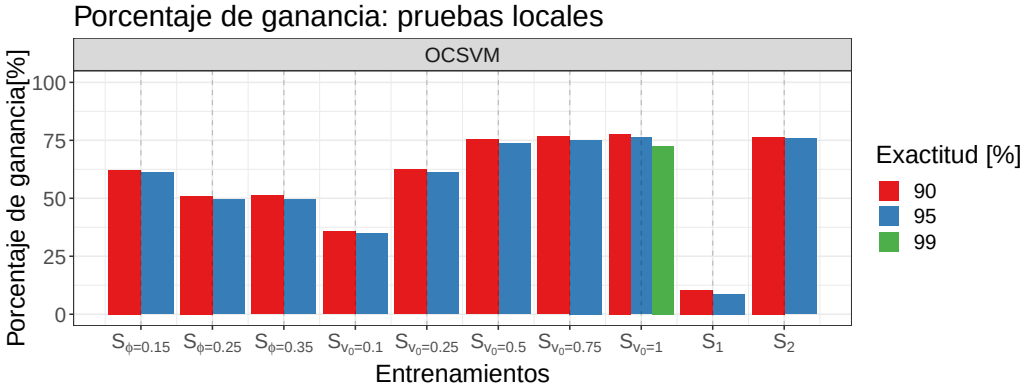

Figura 4.35.: Porcentaje de ganancia temporal de los entrenamientos de *OCSVM* para todas las pruebas locales.

Para el tamaño mediano y grande, los porcentajes de ganancia son los mostrados en las Fig. 4.32 y Fig.4.34. Para el tamaño mediano, el mayor porcentaje de ganancia lo posee el entrenamiento $\mathcal{S}_2$ y el menor nuevamente es el entrenamiento $\mathcal{S}_1$. Para el tamaño grande, $\mathcal{S}_3$ supera los porcentajes de ganancia $\mathcal{S}_2$ por menos de un 1 %.

Si se toma el entrenamiento $\mathcal{S}_2$ y se realiza el análisis completo para las extensiones local y distante, se tiene:

- **Tiempo de etiquetado:** 7.2 segundos (para cada caso individual).
- **Tiempo de entrenamiento:** 0.1 segundos (se realiza una sola vez, es el mismo para los 35 casos).
- **Tiempo de lectura:** 8 horas 25 minutos (totalizado sobre los 100 archivos de cada caso en los tres tamaños).
- **Tiempo consumido T total:** para un 95 % de exactitud, 16 días 8 horas y 19 minutos para todos los tamaños.
- T_n **total de las 35 simulaciones:** 62 días 15 horas y 30 minutos.

El tiempo total demorado (T consumido + Tiempo de lectura) para obtener todas las predicciones mediante $\mathcal{S}_2$ resulta de 16 días, 16 horas y 44 minutos. Por lo tanto, el porcentaje de ahorro temporal total es de 73.3 %.

En conclusión, el sistema de AA desarrollado para realizar esta tarea de clasificación mediante la utilización de *OCSVM* permite obtener el desenlace de 35 simulaciones de distintos parámetros generales con tiempo de simulación real de ∼62 días en tan solo 16 días. En este tiempo el analista es capaz de determinar si las simulaciones acaban en una fragmentación o aglomeración con un 95 % de exactitud.

Se detalla a continuación el ahorro total en términos de espacio ocupado por los archivos de salida de cada simulación:

Tamaño	Espacio ocupado	ϕ	Espacio ahorrado	% de espacio ahorrado
Peq.	42GB	0.15	6.2GB	∼ 63.7 %
		0.25	12GB	
		0.35	8.5GB	
Med.	408GB	0.15	60.2GB	∼ 49.6 %
		0.25	97.8GB	
		0.35	44.2GB	
Gr.	225GB	0.15	111.8GB	∼ 49.7 %

Tabla 4.17: Espacio de almacenamiento ahorrado por el algoritmo *OCSVM* mediante el entrenamiento $\mathcal{S}_2$. LOs valores completos de las simulaciones se encuentran detallados en el **Apéndice B**.

4.7 Estimación de tiempo de corte

Se mencionó en el capítulo anterior cómo el analista generalmente ejecuta las simulaciones una cantidad de pasos de más porque no sabe a priori en cual paso los fragmentos estarán separados totalmente. El programa utilizado en la Sección 3.3.3 para etiquetar y finalizar las 15 simulaciones de entrenamiento en un cierto paso con un criterio uniforme necesita las 15 simulaciones terminadas (los dos clusters separa-

dos). Este programa, por lo tanto, no sirve para aplicar el criterio de corte utilizado en el programa en simulaciones nuevas, es decir, con otros parámetros generales.

Sin embargo, se podría sugerir al analista el paso de finalización de simulaciones para nuevos v_0 y ϕ mediante una extrapolación de los resultados que se poseen, como se especifica a continuación.

En la Fig. 4.36 se muestra el paso de corte de las simulaciones por ϕ, para las cinco v_0 trabajadas. De esta imagen se observa que la dependencia del paso temporal de una cierta $\vec{v}_0$ disminuye su dependencia con ϕ a medida que aumenta su módulo y por ello se realizó un ajuste lineal con una pendiente de cero, o sea, se ajustó solo la ordenada al origen.

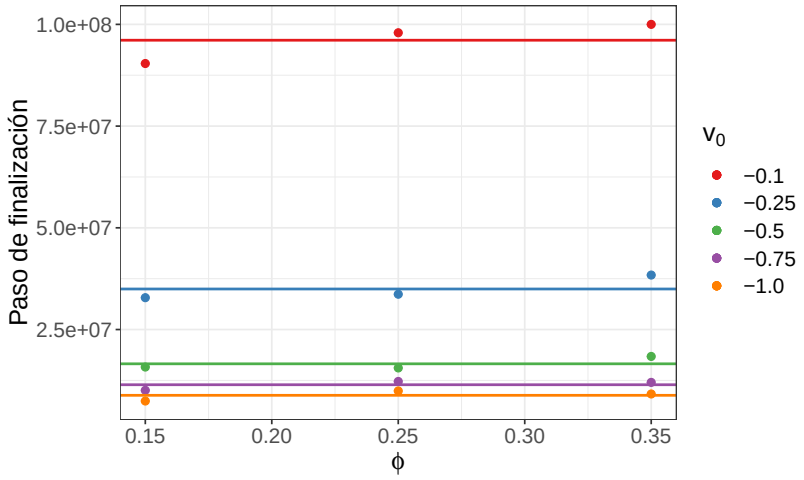

Figura 4.36.: Últimos pasos de los 15 casos de las simulaciones de entrenamiento ajustados mediante una recta horizontal, por ϕ.

Si se quiere simular un caso con alguna v_0 de las que se poseen pero un nuevo ϕ, por ejemplo 0.2 ó 0.3, aproximadamente se le sugiere al analista que corte la simulación en los pasos indicados por las rectas horizontales de la Fig. 4.36. Se ilustran dos ejemplos de la configuración de la simulación en el paso sugerido al analista mediante este ajuste.

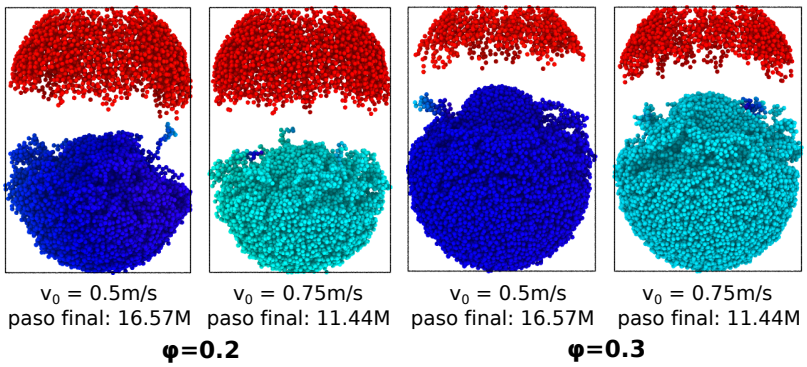

Figura 4.37.: Pasos finales sugeridos al analista por ajuste mostrado en la Fig. 4.36, para $\phi = 0.2$ y 0.3, y $v_0 = 0.5\frac{m}{s}$ y $0.75\frac{m}{s}$.

Como puede observarse del resultado anterior, la estimación es bastante acertada: visualmente los clusters están bien separados, pero no excesivamente alejados uno del otro.

También el analista puede desear ejecutar simulaciones con nuevas velocidades, pero con alguno de los tres ϕ de las simulaciones de entrenamiento. Se aplicó una transformación en la variable $v_0 \to \frac{1}{v_0}$ y se ajustaron los datos agrupados por factor de llenado ϕ mediante una recta, obteniendo:

$$f_{\phi=0.15}\left(\frac{1}{v_0}\right) = (9764320 \pm 214862)\frac{1}{v_0}, \tag{4.4}$$

$$f_{\phi=0.25}\left(\frac{1}{v_0}\right) = (9764320 \pm 214862)\frac{1}{v_0}, \tag{4.5}$$

$$f_{\phi=0.35}\left(\frac{1}{v_0}\right) = (9764320 \pm 214862)\frac{1}{v_0}. \tag{4.6}$$

Se forzó a la ordenada al origen a ser nula, puesto que el ajuste debe describir una situación física realista. El resumen de estos tres ajustes se muestra en la siguiente tabla:

Función	Pendiente	Error Res. Estdr.	R^2	F
$f_{\phi=0.15}$	(8857656 ± 195979)	2172000	0.998	2043
$f_{\phi=0.25}$	(9541385 ± 282809)	3134000	0.9965	1138
$f_{\phi=0.35}$	(9900109 ± 114298)	1266000	0.9995	7502

Tabla 4.18: Resumen estadístico de los ajustes de los pasos finales de las simulaciones de entrenamiento agrupados por ϕ. Se muestran los errores residuales estándar (Error Res. Estdr.), los coeficientes de determinación R^2 y la estadística F.

Las rectas ajustadas se muestran en la Fig. 4.38.

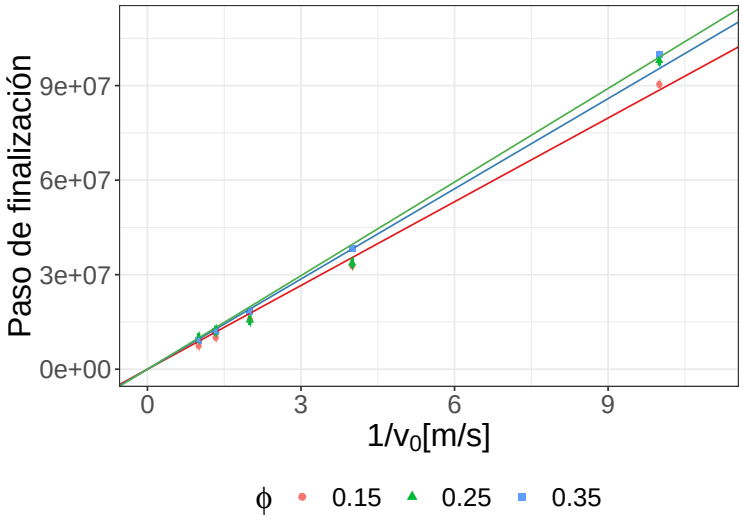

Figura 4.38.: Pasos de finalización de simulación graficados por velocidad.

A partir de estos ajustes, se puede estimar el paso de finalización de nuevos casos, como por ejemplo $(0.4\frac{m}{s}, 0.15)$, $(0.65\frac{m}{s}, 0.15)$ y $(0.85\frac{m}{s}, 0.15)$ mediante $f_{\phi=0.15}$:

	$(0.4\frac{m}{s}, 0.15)$	$(0.65\frac{m}{s}, 0.15)$	$(0.85\frac{m}{s}, 0.15)$
$f_{\phi=0.15}$	(22144140 ± 2172000)	(13627163 ± 2172000)	(10420772 ± 2172000)

Tabla 4.19: Estimación de los pasos de finalización de nuevos casos mediante el ajuste $f_{\phi=0.15}$.

Al no poseer todos los pasos temporales de las simulaciones puesto que los archivos de salida se obtienen cada 1000 pasos, se muestra a continuación (Fig. 4.39) la configuración de los pasos estimados por el ajuste $f_{\phi=0.15}$ de forma aproximada al múltiplo de 1000 más cercano. Tal como puede apreciarse, esta estimación es bastante acertada, ya que los clusters están claramente separados. Al estar estos demasiado alejados entre sí, se recomienda al analista usar las estimaciones del ajuste $f_{\phi=0.15}$ como una cota superior para elegir el paso de finalización de una simulación.

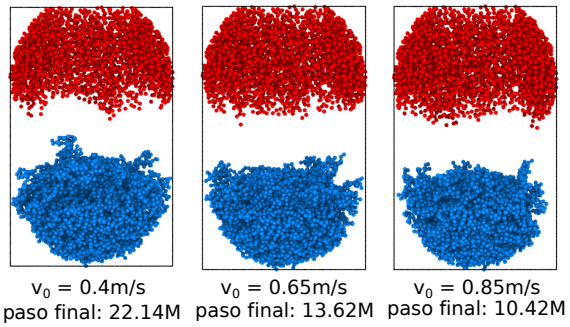

Figura 4.39.: Configuración final de simulaciones con $\phi = 0.15$ con paso de finalización estimado mediante la Ec. 4.4, aproximado al múltiplo de 1000 más cercano.

También se realizó un ajuste lineal despreciando la dependencia del paso final de las simulaciones con ϕ. El resultado del ajuste es

$$f\left(\frac{1}{v_0}\right) = (9433050 \pm 160858)\frac{1}{v_0}, \tag{4.7}$$

como se muestra en la figura 4.40.

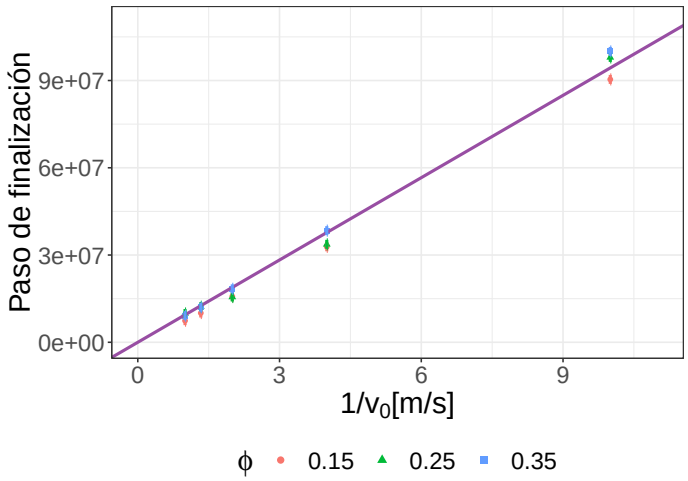

Figura 4.40.: Ajuste lineal $f\left(\frac{1}{v_0}\right)$ para la estimación del paso de finalización de las simulaciones de entrenamiento, independiente del valor de ϕ.

Los valores del ajuste lineal se detallan en la siguiente tabla:

Pendiente	Error Res. Estdr.	R^2	F
(9433050 ± 160858)	3087000	0.9959	3439

Tabla 4.20: Resumen estadístico del ajuste de los pasos finales de las simulaciones de entrenamiento agrupados por ϕ. Se muestran el errore estándar residual (Error Estdr. Res.), el coeficiente de determinación R^2 y la estadística F.

A partir de los valores dados por el ajuste, se tiene que los pasos finales para los casos ejemplificados en la figura 4.39 $(0.4\frac{m}{s}, 0.15)$, $(0.65\frac{m}{s}, 0.15)$ y $(0.85\frac{m}{s}, 0.15)$ son 23582625 ± 3087000, 14512385 ± 3087000 y 11097706 ± 3087000 respectivamente. Evidentemente, los cuales son parecidos a los valores obtenidos previamente, aunque el error residual es bastante alto. Además, como no se toma en cuenta la dependencia del paso final respecto de ϕ, se estima que los pasos de finalización para los casos $(0.5\frac{m}{s}, \phi)$ y $(0.75\frac{m}{s}, \phi)$ resultan 18866100 ± 3087000 y 12577400 ± 3087000 respectivamente.

Se concluye que la independencia de ϕ para los tres ejemplos ilustrados implica no solamente más error residual estándar en las predicciones, sino también pasos de

finalización en los cuales los clusters estarían demasiado separados como para ser esta una estimación óptima.

5. CONCLUSIONES

5.1 Objetivos cumplidos

En este trabajo se desarrolló un método computacional mediante el uso de un sistema de AA para reducir el tiempo de simulaciones de colisiones entre dos granos porosos. Mediante este sistema se realizó una tarea de clasificación de las partículas constituyentes de los granos de acuerdo a su pertenencia al fragmento superior o inferior resultantes de la colisión. No sólo se utilizaron algoritmos supervisados para la tarea principal, también se utilizó el algoritmo no supervisado *k-means* para etiquetar de forma correcta los datos del dominio, es decir, las N partículas del paso final de las simulaciones de entrenamiento. El algoritmo *k-means* funcionó bien para los entrenamientos utilizados (simulaciones de tamaño pequeño) debido a que los clusters quedaron bien delimitados en dos fragmentos, sin valores anómalos. Sin embargo, *k-means* falló en el etiquetado de las configuraciones finales de las simulaciones de tamaño mediano y grande para evaluar las predicciones de los algoritmos supervisados, y tuvo que ser reemplazado por el algoritmo de Aglomeración de Clusters; más preciso, pero computacional y temporalmente más costoso si se lo implementa sin ninguna optimización. Si el analista desease entrenar con conjuntos tomados de simulaciones de estos tamaños de proyectil/blanco mediano y grande, deberá usar este algoritmo, con la desventaja de que el tiempo de etiquetado de los conjuntos de entrenamiento será mayor en comparación con *k-means*. Sin embargo, el tiempo de cómputo de etiquetas en sí con el algoritmo de Aglomeración dista mucho de ser irracional, y su uso es justificado.

Se probaron tres algoritmos para la tarea de clasificación, y ninguno presentó subajuste de los datos. En general, observando la métrica de exactitud, el mejor desempeño en las pruebas desarrolladas lo posee *OCSVM*. Los algoritmos *SVM* y *RF*

no alcanzaron a superar el 90 % de exactitud en todas las pruebas, debido en parte a la forma en la que operan; los predictores resultantes de los entrenamientos con centroides de clusters muy separados fallaban en generalizar a casos cuyos clusters finales poseyesen velocidades promedio finales en z muy próximas. Este tipo de desempeños resultaron en un sobreajuste de los predictores sobre los datos de evaluación.

En los tres algoritmos hubieron sobreajustes manifestados en una pequeña disminución de la métrica de sensibilidad en los pasos finales, pero tal sobreajuste no influyó en la métrica de exactitud de forma significativa. En las pruebas en las que *SVM* y *RF* se desempeñaron correctamente en todos los casos evaluados, casi siempre el tiempo ahorrado por estos algoritmos fue mayor que el de *OCSVM*.

OCSVM fue el algoritmo cuyo funcionamiento básico resultó ser el más adecuado para lograr la clasificación este tipo de simulaciones. Logró desempeñarse bien en todas las pruebas realizadas, tanto para la extensión local como la distante. El único problema que posee *OCSVM* es que sus parámetros no se generalizan de manera perfecta para los casos con velocidades altas, presentando un sobreajuste manifestado en el decaimiento de la exactitud en los últimos pasos de simulación, como se mencionó anteriormente. Sin embargo, este sobreajuste no impide que la exactitud supere el 95 % en todos los casos, y por ende no se considera significativo comparado con los sobreajustes de *RF* y *SVM*, los cuales no logran siquiera superar el 80 % de exactitud en algunos casos de este tamaño.

Tomando en cuenta los tiempos de etiquetado, entrenamiento de algoritmo, lectura de archivos y resultados de predicción, *OCSVM* logra predecir el desenlace de 35 simulaciones (de distintos tamaños) con un 95 % de exactitud ahorrando un 73.3 % del tiempo que demoraría correr todas esas simulaciones hasta el final.

Con este resultado se muestra que una simulación de tamaño pequeño, velocidad inicial de proyectil de $0.1\frac{m}{s}$ y factor de llenado $\phi = 0.35$ es suficiente para predecir el desenlace de simulaciones de otros tamaños, velocidades y factores de llenado. Además, esta clasificación no se lleva a cabo en el último paso de cada simulación, sino en un paso mucho anterior. Esto es indicio de que no solamente la distribu-

ción generadora de la configuración final de un solo caso y tamaño da cuenta de las configuraciones finales de otras simulaciones con parámetros distintos, sino también que genera configuraciones de pasos anteriores de las mismas. Cabe destacar el espacio computacional ahorrado mediante este entrenamiento de *OCSVM*: en total, de 675GB de espacio ocupado por los archivos de salida de todas las simulaciones, sólo se utilizan $\sim$ 334.3GB, es decir, con casi el 50 % menos de información se puede obtener el desenlace de las colisiones simuladas con un 95 % de exactitud. Además, esto implica intrínsecamente un ahorro energético considerable, y es otra de las ventajas que conlleva la práctica del método propuesto.

Debido la forma del problema tratado en este Seminario y las características de operación de *OCSVM*, era esperable este buen desempeño por parte del algoritmo. Dado lo adecuado que resulta *OCSVM* para realizar la clasificación, ¿es necesario entonces utilizar *SVM* y *RF*? Como aclaramos en la Sección 2.3.1, por el teorema de NFL, no hay manera de elegir a priori un algoritmo universal para realizar correctamente ciertas tareas, se debe poseer un criterio externo al algoritmo para poder elegirlo correctamente. En el caso tratado, *OCSVM* resulta el más indicado para realizar la tarea. Sin embargo, *RF* y *SVM* poseen una característica que *OCSVM* no posee, y es la posibilidad de extender su capacidad de clasificación binaria a lo que se denomina "clasificación multiclase" o *multi-class classification*, la cual consiste en catalogar datos en tres o más etiquetas. Si se da un escenario en el cual nuevos parámetros de este tipo de simulaciones de colisiones devengan en más de dos fragmentos, *OCSVM* resultaría obsoleto, pero no así *RF* y *SVM*, y su utilización para clasificar este tipo de colisiones no sería incoherente.

Por último, se presentó mediante ajustes lineales de datos para los pasos finales de las simulaciones pequeñas una sugerencia para el analista sobre el paso de finalización de nuevas simulaciones, es decir, con valores de ϕ y v_0 distintos a los utilizados en las simulaciones de entrenamiento. Los resultados de los ejemplos realizados muestran que es un criterio de corte de finalización bastante acertado, lo cual indica una correlación entre los pasos de finalización de una simulación y sus parámetros generales.

5.2 Propuestas futuras

La ventaja de los sistemas de AA es la gran cantidad de variables y parámetros que pueden modificarse para mejorar los resultados. Además, cada parte del sistema ofrece nuevos caminos de exploración. Como trabajo futuro se propone:

- Se podría explorar qué tan factible es predecir el desenlace de colisiones dentro de un mismo tamaño de simulación y caso, pero variando las condiciones iniciales del sistema (rotar los granos, armar varias muestras con disposiciones de partículas al azar, etc.)
- Se deberían analizar los quince casos de tamaño grande, esto no se pudo realizar porque no se disponía de las simulaciones.
- En definitiva, entrenar un algoritmo con un cierto $\mathcal{S}$ y luego utilizarlo para predecir sobre otro conjunto de datos diferente es la base de *Transfer Learning* o aprendizaje transferido. Este método busca encontrar similitudes entre distribuciones generadoras de datos. Para que el trabajo aquí realizado sea considerado aprendizaje transferido, se debe encontrar una manera de escalar los parámetros de los entrenamientos, adecuándolos a cada uno de los casos y tamaños probados. De esta forma, un mismo entrenamiento cambia sus parámetros según las características de los datos sobre los que deba predecir, de forma automática. En este trabajo se fijaron los parámetros al entrenar los algoritmos y se testeaban los mismos sobre todos los casos. Esta rigidez puede ser la causante del sobreajuste de los datos.
- Se podrían mejorar los programas realizados de tal manera que clasifiquen si la colisión deviene en una fragmentación o aglomeración de manera automática. Esto no se realizó debido a que el criterio de cantidad de partículas en cada fragmento para catalogar la colisión como una u otra es ambiguo y subjetivo al analista.
- Para disminuir aún más el tiempo de simulación, se propone investigar reducir la cantidad de partículas sobre las que se trabajan en cada simulación mediante

un muestreo, evaluando por ejemplo, la exactitud de los algoritmos entrenados con solamente un 10 %, 20 %, 30 %, etc., de los datos constituyentes de las simulaciones de entrenamiento.

- Para evitar las superposiciones en los entrenamientos con los cuales RF y SVM tienen un mal desempeño, se puede normalizar la variable v_z mediante v_0 de cada caso. De esta forma se obtienen entrenamientos linealmente separados, y el desempeño de los algoritmos mejora considerablemente. Se probó esto en un caso conflictivo, pero se encontró que $OCSVM$ tiene un mal desempeño producto de un sobreajuste, y se necesita más tiempo de investigación para comprender la causa.

LISTA DE REFERENCIAS

LISTA DE REFERENCIAS

[1] D. Glazer, R. Radmer, and R. Altman, "Combining molecular dynamics and machine learning to improve protein function recognition," *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing.*, p. 332, 2008.

[2] M. Ziatdinov, O. Dyck, A. Maksov, X. Li, X. Sang, K. Xiao, R. Unocic, R. Vasudevan, S. Jesse, and S. Kalinin, "Deep learning of atomically resolved scanning transmission electron microscopy images: Chemical identification and tracking local transformations," *ACS Nano*, vol. 11, p. 1274212752, 2017.

[3] Z. Shi, E. Tsymbalov, M. Dao, S. Suresh, A. Shapeev, and J. Li, "Deep elastic strain engineering of bandgap through machine learning," *Proceedings of the National Academy of Sciences*, vol. 116, pp. 4117–4122, 2019.

[4] "Materials acceleration platform: Accelerating advanced energy materials discovery by integrating high-throughput methods with artificial intelligence," *Report of the Clean Energy Materials Innovation Challenge Expert Workshop*, 2018.

[5] J. S. Uehara, M. A. Ambroso, R. P. Ojha, and D. J. Durian, "Low-speed impact craters in loose granular media," *Phyical Review Letters*, 2003.

[6] D. Lohse, R. Bergmann, R. Mikkelsen, C. Zeilstra, D. van der Meer, M. Versluis, K. van der Weele, M. van der Hoef, and H. Kuipers, "Impact on soft sand: Void collapse and jet formation," *Phyical Review Letters*, 2004.

[7] M. P. Ciamarra, A. H. Lara, A. T. Lee, D. I. Goldman, I. Vishik, and H. L. Swinney, "Dynamics of drag and force distributions for projectile impact in a granular medium," *Phyical Review Letters*, 2004.

[8] H. Katsuragi, "Morphology scaling of drop impact onto a granular layer," *Phyical Review Letters*, 2010.

[9] C. Dominik and A. G. G. M. Tielens, "The physics of dust coagulation and the structure of dust aggregates in space," *The Astrophysical Journal*, vol. 480, pp. 647–673, 1997.

[10] D. Paszun and C. Dominik, "Collisional evolution of dust aggregates. from compaction to catastrophic destruction," *Astronomy and Astrophysics*, vol. 507, pp. 1023–1040, 2009.

[11] K. Wada, H. Tanaka, T. Suyama, H. Kimura, and T. Yamamoto, "Numerical simulation of dust aggregate collisions. i. compression and disruption of two-dimensional aggregates," *Astrophysics Journal*, vol. 661, p. 320, 2007.

[12] K. Wada, H. Tanaka, T. Suyama, H. Kimura, and T. Yamamoto, "Numerical simulation of dust aggregate collisions. ii. compression and disruption of three-

dimensional aggregates in head-on collisions," *Astrophysics Journal*, vol. 677, p. 1296, 2008.

[13] K. Wada, H. Tanaka, T. Suyama, H. Kimura, and T. Yamamoto, "Collisional growth conditions for dust aggregates," *Astrophysics Journal*, vol. 702, p. 1490, 2009.

[14] K. Wada, H. Tanaka, T. Suyama, H. Kimura, and T. Yamamoto, "The rebound condition of dust aggregates revealed by numerical simulation of their collisions," *Astrophysics Journal*, vol. 737, p. 36, 2011.

[15] C. Ringl and H. M. Urbassek, "A lammps implementation of granular mechanics: inclusion of adhesive and microscopic friction forces," *Computer Physics Comunications*, vol. 183, pp. 986–992, 2012.

[16] J. Blum, "Dust growth in protoplanetary disks — a comprehensive experimental/theoretical approach," *Research in Astronomy and Astrophysics*, vol. 10, 2010.

[17] P. J. Armitage, *"Astrophysics of planet formation", Cambridge University Press.* 2010.

[18] C. Ringl, E. M. Bringa, and H. M. . Urbassek, "Impact on porous targets: Penetration, crater formation, target compaction and ejection," *Physical review*, vol. 86, p. 061313, 2012.

[19] C. Ringl, E. M. Bringa, D. S. Bertoldi, and H. M. . Urbassek, "Collisions of porous clusters: a granular-mechanics study of compaction and fragmentation," *The Astrophysical Journal*, vol. 752, p. 151, 2012.

[20] M. B. Planes, E. N. Millán, H. M. Urbassek, and E. M. Bringa, "Dust-aggregate impact into granular matter: A systematic study of the influence of projectile velocity and size on crater formation and grain ejection," *Astronomy & Astrophysics*, vol. 607, p. A19, 2017.

[21] T. Birnstiel, M. Fang, and A. Johansen, "Dust evolution and the formation of planetesimals," *Space Science Reviews*, vol. 205, pp. 41–75, 2016.

[22] J. Blum, "Experiments on sticking, restructuring, and fragmentation of preplanetary dust aggregates.," *Icarus*, vol. 143, p. 138–146, 2000.

[23] A. Seizinger, R. Speith, and W. Kley, "Compression behavior of porous dust agglomerates.," *Astronomy & Astrophysics*, vol. 541, p. A59, 2012.

[24] A. Whizin, J. Blum, and J. Colwell, "The physics of protoplanetesimal dust agglomerates. viii. microgravity collisions between porous sio_2 aggregates and loosely bound agglomerates.," *The Astrophysical Journal*, vol. 836, p. 94, 2017.

[25] M. B. Planes, E. N. Millán, H. M. Urbassek, and E. M. Bringa, "Fragmentation and fracture of porous aggregates at low collision velocities – a granular mechanics study of mass-asymmetric collisions," *Submitted to: Astronomy Astrophysics*, vol.-, pp.–, subm: February 2019.

[26] E. N. Millán, C. J. Ruestes, N. Wolovick, and E. M. Bringa, "Boosting materials science simulations by high performance computing," *Mecánica Computacional*, vol. XXXV, pp. 467–482, 2017.

[27] D. Glazer, R. Radmer, and R. Altman, "Combining molecular dynamics and machine learning to improve protein function recognition," *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing.*, p. 322, 2008.

[28] M. Rupp, A. Tkatchenko, K. Muller, and O. von Lilienfeld, "Fast and accurate modeling of molecular atomization energies with machine learning," *Physical Review Letters*, vol. 108, 2012.

[29] V. Botu and R. Ramprasad, "Adaptive machine learning framework to accelerate ab initio molecular dynamics," *International Journal of Quantum Chemistry*, vol. 115, p. 1074–1083, 2014.

[30] S. Shalev-Schwartz and S. Ben-David, *"Understanding Machine Learning from theory to algorithms", Cambridge University Press.* 2014.

[31] D. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Computation*, pp. 1341–1390, 1996.

[32] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," *Proceeding COLT '92 Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.

[33] V. Vapnik, *"The nature of statistical learning theory" in Springer, Berlin, Germany.* 1995.

[34] B. Scholkopf, C. Burges, and A. Smola, *"Advances in Kernel Methods: Support Vector Learning," in MIT Press, Cambridge, MA, USA.* 1999.

[35] B. Schlkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson., "Support vector method for novelty detection," *Conference: Advances in neural information processing systems*, pp. 582–588, 2000.

[36] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[37] T. Hastie, R. Tibshirani, and J. Friedman, *"The elements of statistical learning," in Springer.* 2001.

[38] L. Rokach and O. Maimon, *"Clustering methods. Data mining and knowledge discovery handbook," in Springer.* 2005.

[39] A. Kassambara, *"Practical Guide to Cluster Analysis in R " in STHDA.* 2017.

[40] J. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, 1967.

[41] C. Ringl and H. Urbassek, "A simple algorithm for constructing fractal aggregates with pre-determined fractal dimension," *Computer Physics Communications*, p. 1683–1685, 2013.

[42] E. Millán, C. Ringl, C. S. Bederián, M. F. Piccoli, C. G. Garino, H. M. Urbassek, and E. M. Bringa, "A gpu implementation for improved granular simulations with lammps," vol. Garino, C.G., Printista, M. (eds.) VI Latin American Symposium on High Performance Computing HPCLatAm 2013., p. 89–100, 2013.

[43] A. Stukowski, "Visualization and analysis of atomistic simulation data with ovito–the open visualization tool," *Modelling and Simulation in Materials Science and Engineering*, vol. 18, 2009.

[44] M. Kuhn, "The caret package," *R Foundation for Statistical Computing*, 2012.

[45] E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, A. Weingessel, and M. F. Leisch, "Package 'e1071'.," *R Software package*, 2009.

APÉNDICE

A. ESPECIFICACIONES DE HARDWARE Y SOFTWARE

En este Apéndice se detallan los equipos y los programas utilizados para todo el análisis computacional del presente trabajo.

Se describirán:

- **Ejecución de Simulaciones:** Análisis de equipos y programas para llevar a cabo las simulaciones utilizadas.
- **Herramientas de visualización:** Utilizadas para representar las partículas de simulaciones, las colisiones y las características de los sistemas tratados.
- **Programas de análisis estadístico e implementación de AA:** Programas utilizados para armar el sistema de AA planteado y para el posterior análisis de desempeño de los algoritmos.

A.1 Ejecución de Simulaciones

La siguiente lista detalla las especificaciones de hardware y software para obtener las simulaciones trabajadas. Para armar los programas con la especificación de los parámetros generales de las colisiones, las condiciones iniciales, las interacciones físicas entre partículas, etc. se utilizó el software LAMMPS de MD en GPU [15, 42].

Los equipos utilizados para ejecutar estos programas se detallan a continuación:

- Estación de trabajo FX-8350 con: AMD FX-8350 con 8 núcleos a 4 GHz con 32 GB de memoria RAM DDR3. Con una NVIDIA GeForce GTX Titan X (arquitectura Maxwell GM200) con 12 GB de memoria. Sistema operativo Slackware Linux 14.2 de 64 bits con el kernel 4.4.14, OpenMPI 1.8.4, GCC 5.3.0, versión de lenguaje R 3.5.1 y Cuda 6.5 con el controlador NVIDIA 375.66.

- Estación de trabajo FX-8350 con: las mismas especificaciones que la estación de trabajo anterior pero con una NVIDIA GeForce GTX Titan Xp (arquitectura Pascal GP102) con 12 GB de memoria.
- Cluster Toko en la Universidad Nacional de Cuyo:
  - un nodo con cuatro CPU AMD Opteron 6376, con 16 núcleos de CPU en 2.3GHz (cada uno, 64 núcleos en total), 128 GB de RAM y Gigabit Ethernet.
  - un nodo con dos CPU AMD EPYC 7281, con 16 núcleos de CPU a 2,1 GHz (cada uno, 32 núcleos en total), 128 GB de RAM y Gigabit Ethernet.
  - ambos nodos con Slackware Linux 14.1 64 bit con el kernel 4.4.14, OpenMPI 1.8.8, GCC 4.8.2 y Cuda 6.5 con el controlador NVIDIA 396.26.

Como referencia, las simulaciones en GPU se ejecutan ~ 3.6 veces más rápido que en un núcleo de CPU y ~ 1.6 veces más rápido que 8 núcleos de CPU (NVIDIA Titan Xp en comparación con una CPU AMD EPYC 7281).

A.2 Herramientas de visualización

Para realizar las representaciones visuales de las partículas colisionantes (como por ejemplo, Figs. 3.6, 3.7, 3.8)) se utilizó la herramienta de visualización OVITO (Stukowski 2009) [43].

Para los gráficos de variables (Figs. 4.3, 4.5, 4.9, 4.4 etc.) se utilizó el paquete *ggplot2* de R.

Por último, para los esquemas y figuras en general (Figs. 2.2, 3.12, 3.10, etc.) se utilizó el programa de dibujo de código abierto *Inkscape*.

A.3 Programas de análisis estadístico e implementación de AA

Para llevar a cabo los análisis estadísticos y para desarrollar los programas de procesamiento de datos para implementar los algoritmos de AA, se utilizó principalmente el lenguaje R. Los principales paquetes de R utilizados fueron:

- *dplyr* y *plyr* para la manipulación de datos.
- *caret*, *e1071* para los algoritmos supervisados de AA.
- *factoextra*, *parallelDist* y *fastcluster* para los algoritmos no supervisados.
- *purrr* para programación funcional de listas, a través de la función *map()*.
- *tictoc* para medir fácilmente el tiempo de cálculo de las secciones del código R.
- *RColorBrewer* para modificar la parte estética de los gráficos.

Para realizar los análisis y desarrollar los programas de cada prueba presentada en el trabajo se utilizó principalmente una notebook con un procesador Intel Core i7-7500U de 2.70GHz con 4 núcleos y 8 GB de memoria RAM, con sistema operativo Kubuntu 17.10 de 64 bits con el kernel 4.13.0. Para ejecución de pruebas computacionalmente costosas se utilizó una de las estaciones FX-8350 y un nodo del cluster Toko, especificados anteriormente.

B. TAMAÑO DE LAS SIMULACIONES Y ARCHIVOS DE SALIDA

Tanto para entrenar como para evaluar el desempeño de los algoritmos, se utilizaron un total de 35 simulaciones: 15 de tamaño pequeño, 15 de tamaño mediano y 5 de tamaño grande. Cada simulación individual posee cierta cantidad de archivos de salida con un cierto tamaño.

Para las simulaciones de tamaño proyectil/blanco pequeño (entre 11200 y 26206 partículas), el tamaño de cada archivo de salida comprimido (gzip) está entre aproximadamente $400KB$-$1000KB$ (factor de llenado de 0.15-0.35). El número de archivos de salida generados varía entre las diferentes configuraciones simuladas. Las simulaciones con velocidades mayores (por ejemplo $1\frac{m}{s}$) ejecutaron menos pasos totales que las simulaciones con velocidades menores ($0.1\frac{m}{s}$) pues necesitaron más pasos totales para producir la división esperada entre fragmentos. El tamaño total de cada simulación comprimida varía de aproximadamente $1GB$ (factor de llenado 0.15 para $1\frac{m}{s}$) a $8.5GB$ (factor de llenado 0.35 para $0.1\frac{m}{s}$) con un total de $42GB$ de datos comprimidos considerando las 15 simulaciones de este tamaño.

Sin embargo, no se utilizó el volumen de datos detallados anteriormente, solamente un subconjunto de 100 archivos de salida de cada caso. En total, las 15 simulaciones de este tamaño utilizadas ocupan un espacio comprimido de aproximadamente $1.1GB$.

Para las simulaciones de tamaño proyectil/blanco mediano (entre 31087 y 72353 partículas), el tamaño de cada archivo de salida es de aproximadamente $800KB$-$3MB$ (factor de llenado de 0.15-0.35). La cantidad de pasos en cada caso también varía según lo explicado en el párrafo anterior. El tamaño total comprimido de las 15 simulaciones con la submuestra de 100 archivos de salida en cada uno, ocupa un espacio total de $3.1GB$.

Finalmente, en las 5 simulaciones de tamaño de proyectil/blanco grande (entre 51888 y 120894 partículas), el tamaño de cada archivo de salida es de aproximadamente $1.4MB$-$2.5MB$ (factor de llenado de 0.15-0.35). El tamaño total comprimido de las 5 simulaciones de este tamaño, con la submuestra de 100 archivos de salida en cada uno, ocupa un espacio total de $1.1GB$.

C. DETERMINACIÓN DE LA VARIABLE PREDICTORA EN LOS ALGORITMOS SUPERVISADOS

Se mencionó en la sección 3.3.2 que la velocidad en z de las partículas del sistema colisionante del tipo tratado resulta la más indicada como variable predictora. Las pruebas aquí analizadas y todo el desarrollo presentado se realizó con los algoritmos de aprendizaje supervisado entrenados con esta variable predictora.

La elección de esta variable no es trivial, requiere una exploración de los datos obtenidos de los archivos de salida de cada simulación para hallar la que mejor represente la estructura fragmentada final de las colisiones. También cabe destacar que no necesariamente la variable predictora es única, pueden ser varias o alguna combinación matemática de ellas. Para la determinación de tal variable (o variables), principalmente se utilizó las herramientas ofrecidas por la herramienta de visualización OVITO.

Lo primero a observar son las características generales principales de las colisiones simuladas:

1. **El parámetro de impacto:** Estas colisiones son centrales, el proyectil se mueve con una velocidad $\vec{v_0}$ vertical hacia abajo, hacia el centro de masa del blanco. Una vez que atraviesa el blanco, el proyectil y las partículas arrastradas por él continúan una trayectoria vertical hacia abajo.

2. **La velocidad inicial del sistema:** El proyectil es el que posee v_0, el blanco siempre está inicialmente en reposo.

3. **El desenlace de las simulaciones:** Las simulaciones trabajadas poseen la misma configuración final; solo dos fragmentos resultantes, donde el fragmento que queda arriba será etiquetado como cluster 1 y el que es arrastrado hacia abajo será el cluster 2.

4. **El comportamiento general:** Ninguna partícula del proyectil será parte del cluster 1, solamente lo serán algunas del blanco.

5. **La velocidad final promedio de los fragmentos:** Siempre el cluster 1 tendrá una velocidad promedio final cercana a cero, y el cluster 2 tendrá una velocidad considerablemente distinta al cluster 1.

Se poseen en total 11 variables en cada archivo de salida que determinan la configuración del sistema: **id** (número de identificación de la partícula, fijo para todos los pasos temporales), el **tipo de partícula** (1 ó 2 dependiendo si en el paso temporal 0 la partícula es parte del blanco o el proyectil), las **posiciones en x, y, z**, las **velocidades en x, y, z** y las **velocidades angulares en x, y, z**. Las variables **id** y **tipo de partícula** fueron descartadas como variables predictoras por razones obvias. Se necesita al menos una variable predictora capaz de reconocer los fragmentos finales (o la mayoría de la partículas que pertenecerán a ellos) no solamente en el tiempo final de la colisión, sino también en un tiempo anterior. Se comenzó probando las variables de forma individual, como se muestra en la Fig. C.1.

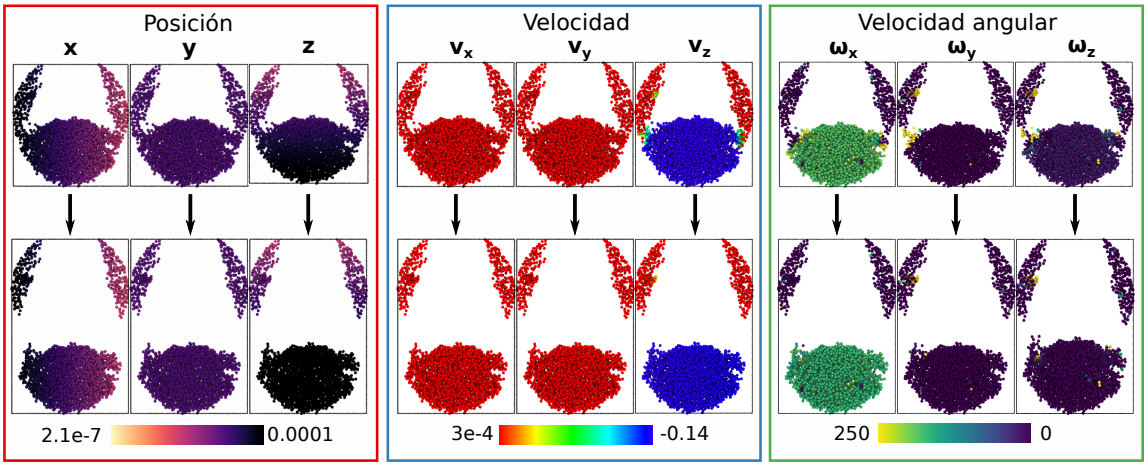

Figura C.1.: Se muestran las partículas colisionantes en dos pasos temporales distintos, coloreados por posición (x, y, z), velocidad (v_x, v_y, v_z) y velocidad angular (ω_x, ω_y, ω_z).

De la Fig. C.1 se observa que las únicas variables que dividen las partículas (o la mayoría de ellas) en los dos clusters finales son la velocidad en z, la velocidad angular

en x y la posición en z. La velocidad angular fue descartada como predictora por la gran variación de sus valores dentro de cada cluster en cada paso temporal.

Se ejecutaron algunas pruebas con la velocidad en z y la posición en z como variables predictoras. Los resultados de desempeño fueron del tipo mostrado en la Fig. C.2, donde los algoritmos no alcanzan siquiera el 90 % de exactitud, para el entrenamiento $\mathcal{S}_{\phi=0.35}$.

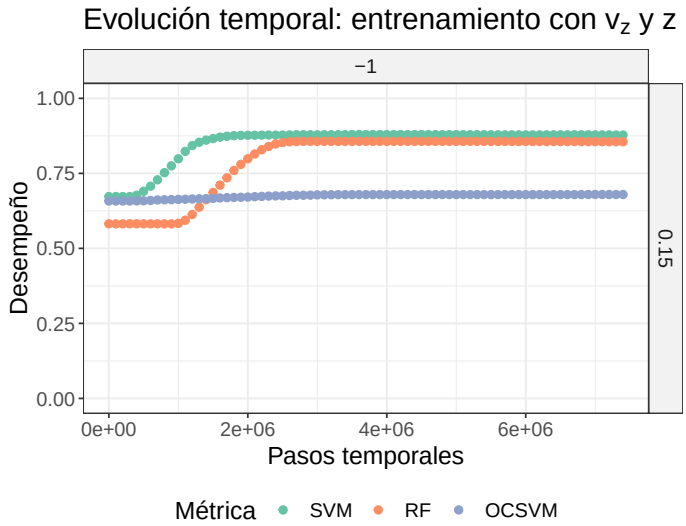

Figura C.2.: Evolución de la métrica **exactitud** en los tres algoritmos supervisados evaluados, entrenados con el entrenamiento $\mathcal{S}_{\phi=0.35}$ con variables predictoras v_z y z, evaluados en el caso $(1\frac{m}{s}, 0.15)$.

Como la posición en z resultó ser la variable que introdujo más error a la clasificación (Fig. C.1), se limitó a utilizar la variable de velocidad en z como predictora.

Se probó además de la velocidad en z, combinaciones de variables sugeridas por la bibliografía, como por ejemplo la velocidad lateral v_l [25], la cual es la componente perpendicular a la velocidad de impacto:

$$v_l = \sqrt{v_x^2 + v_y^2}. \tag{C.1}$$

Los resultados de usar esta combinación fueron iguales a los de entrenar a los algoritmos solo con v_z, pues como muestra la figura C.1, v_x y v_y varían muy poco a lo

largo de la colisión y la velocidad lateral es casi nula. Sin embargo, se descartó esta combinación de variables predictoras pues utilizar más variables en el entrenamiento implica mayor tiempo de cómputo.

D. ESPECIFICACIÓN DE HIPERPARÁMETROS UTILIZADOS

Se especifica a continuación los hiperparámetros utilizados para cada algoritmo (*RF*, *SVM* y *OCSVM*) en los 11 entrenamientos.

- **Entrenamientos $\mathcal{S}_{\phi=*}$**

		Entrenamientos		
Algoritmo	**Hiperparámetro**	$\mathcal{S}_{\phi=0.15}$	$\mathcal{S}_{\phi=0.25}$	$\mathcal{S}_{\phi=*}$
RF	*mtry*	2	2	2
SVM	C	1	0.5	1
	σ	12395.1	6232.365	-
OCSVM	ν	0.01	0.01	0.01
	γ	0.001	0.001	0.001

Tabla D.1: Se muestran los hiperparámetros con mejor exactitud resultantes de la validación cruzada de la grilla provista a los algoritmos para los entrenamientos por ϕ. Los valores de σ de *SVM* son sólo para los casos en los cuales se entrena con el núcleo radial.

- **Entrenamientos $\mathcal{S}_{v_0=*}$**

		Entrenamientos				
Algoritmo	**Hiperparámetro**	$\mathcal{S}_{v_0=0.1}$	$\mathcal{S}_{v_0=0.25}$	$\mathcal{S}_{v_0=0.5}$	$\mathcal{S}_{v_0=0.75}$	$\mathcal{S}_{v_0=1}$
RF	*mtry*	2	2	2	2	2
SVM	C	1	1	1	1	1
	σ	-	-	-	-	-
OCSVM	ν	0.01	0.01	0.01	0.01	0.01
	γ	0.001	0.001	0.001	0.001	0.001

Tabla D.2: Se muestran los hiperparámetros con mejor exactitud resultantes de la validación cruzada de la grilla provista a los algoritmos para los entrenamientos por v_0. Los valores de σ de *SVM* son sólo para los casos en los cuales se entrena con el núcleo radial.

- **Entrenamientos combinados $\mathcal{S}_*$**

		Entrenamientos		
Algoritmo	**Hiperparámetro**	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$
RF	*mtry*	2	2	2
SVM	C	1	1	1
	σ	-	-	-
OCSVM	ν	0.01	0.01	0.01
	γ	0.001	0.001	0.001

Tabla D.3: Se muestran los hiperparámetros con mejor exactitud resultantes de la validación cruzada de la grilla provista a los algoritmos para los entrenamientos combinados. Los valores de σ de *SVM* son sólo para los casos en los cuales se entrena con el núcleo radial.

Los programas completos utilizados en el Seminario se encuentran en `https://github.com/RimDan/script_colisiones`.