UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

HERRAMIENTAS Y BANCO DE PRUEBAS PARA LA DETECCIÓN DE ANOMALÍAS EN FLUJOS DE DATOS

HE CHEN

Director/a: CONRADO MARTÍNEZ PARRA (Departamento de Ciencias de la Computación)

Titulación: Grado en Ingeniería Informática (Computación)

Memoria del trabajo de fin de grado

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

22/01/2024

Agradecimientos

Quisiera expresar mi agradecimiento a todas las personas que, de una forma u otra, han contribuido a la realización de este proyecto.

En primer lugar, quiero agradecer especialmente a mi director de proyecto, por su guía y apoyo a lo largo de todo el proceso. Sus consejos y orientaciones han sido fundamentales para el desarrollo de este trabajo.

También extiendo mi gratitud a los autores de los trabajos e investigaciones que he utilizado como referencia. Su disposición a compartir sus conocimientos y resultados ha sido esencial para el éxito de este proyecto.

Un agradecimiento a mis compañeros, amigos y familiares, por su apoyo y comprensión durante los momentos más desafiantes de este viaje académico.

¡Muchas gracias!

Abstract

Abstract - Castellano

Este proyecto final se centra en detectar anomalías en el comportamiento de ítems específicos dentro de un flujo de datos, considerando su comportamiento pasado. Debido a limitaciones en la memoria disponible, no es posible desarrollar modelos independientes para cada ítem distinto en el flujo de datos. Nuestro proyecto tiene como objetivo proponer algoritmos y herramientas para abordar este problema, haciéndolos disponibles en una biblioteca de software lista para usar, y realizar experimentos para evaluar su rendimiento y calidad.

Abstract - English

This final project focuses on detecting anomalies in the behavior of specific items within a data stream, considering their past behavior. Due to limitations in available memory, it is not feasible to develop independent models for each distinct item in the data stream. Our project aims to propose algorithms and tools to address this issue, making them available in a ready-to-use software library, and to conduct experiments to assess their performance and quality.

Índice general

Índice de figuras

1 Introducción

Este documento constituye la memoria del Trabajo de Fin de Grado (TFG) para la titulación de Grado en Ingeniería Informática, especialidad en Computación, desarrollado en la Facultad de Informática de Barcelona (FIB) de la Universidad Politécnica de Cataluña (UPC). El proyecto ha sido conducido bajo el apoyo y orientación del director Conrado Martínez Parra, perteneciente al Departamento de Ciencias de la Computación.

El propósito de este proyecto está en la detección y análisis de anomalías en el comportamiento de ítems específicos dentro de *data streams*, considerando su historial y patrones pasados. Dada la restricción en la capacidad de memoria y la imposibilidad de generar modelos individuales para cada ítem en el flujo de datos, nuestro trabajo se orienta hacia la identificación y aplicación de algoritmos y herramientas existentes capaces de superar tales limitaciones. El fin último es integrar estas soluciones en una biblioteca de software de fácil acceso y uso, además de conducir una serie de experimentos para evaluar su eficacia y desempeño.

Es crucial subrayar que el enfoque del proyecto no es la invención de nuevos algoritmos, sino la exploración, adaptación y optimización de técnicas y herramientas preexistentes. En otras palabras, buscamos emplear y potenciar si es posible lo que ya ha sido desarrollado en el campo, adaptándolo a nuestras necesidades

específicas y contribuyendo al ámbito de la detección de anomalías con un recurso práctico y robusto.

Debido al carácter investigativo de este proyecto, se enfrenta a un riesgo significativo asociado principalmente a la falta inicial de conocimiento profundo sobre el campo de estudio. Este riesgo se manifiesta en una prolongada etapa de aprendizaje que constituye la base del trabajo. Dicha etapa de aprendizaje abarca una serie de tareas críticas, incluyendo la lectura y análisis de artículos de investigación (*papers*), el examen de alternativas, la comprensión detallada de la solución seleccionada y el dominio de las herramientas necesarias para implementar o emplear dicha solución.

Dado este contexto, es posible que se requieran ajustes menores o rectificaciones de los objetivos iniciales. Esto puede suceder si, durante la fase de aprendizaje, se detecta que la ruta planificada no es viable o si se descubre un enfoque alternativo más prometedor y adecuado. Estos cambios, lejos de ser un retroceso, son una adaptación necesaria y una respuesta proactiva a las complejidades y descubrimientos a la investigación.

En la práctica, después de un análisis detallado del estado del arte en el campo de la detección de anomalías, identificamos dos soluciones que se ajustan mejor a nuestros requisitos definidos. Estos dos algoritmos son el Sparse Data Observer (SDO) y SENCForest. Realizamos un estudio del funcionamiento de ambos para decidir cuál profundizar. Finalmente, optamos por SENCForest. Esta elección refleja el carácter investigativo del proyecto, que nos llevó a ajustar los objetivos iniciales tras un análisis detallado del campo.

Nuestras contribuciones incluyen la implementación de un generador de datos sintéticos diseñado para el contexto de detección de novedades. Este generador es una versión modificada del empleado por los autores de SDO y ha sido utilizado en los experimentos con SENCForest. Además, implementamos las métricas de evaluación para analizar el desempeño del algoritmo. Estas métricas, aunque definidas en el artículo original del algoritmo, no estaban implementadas en el código fuente proporcionado por el autor. Estas contribuciones están documentadas y disponibles para revisión o experimentación en la página de GitHub del proyecto:

www.github.com/IDBIDO/TFG, también estará en subido como *Material adicional* de la memoria del proyecto.

En las próximas secciones, introduciremos el contexto y los conceptos clave relevantes al proyecto. Seguidamente, abordaremos el problema inicialmente planteado, que servirá como modelo para guiarnos en este proceso exploratorio. Es posible que, al finalizar, el resultado presente similitudes y pequeñas diferencias con respecto al modelo original. Posteriormente, discutiremos la metodología y la planificación del trabajo, enfocándonos en la gestión del proyecto. Tras detallar estos aspectos, procederemos a explorar la investigación e implementación de la solución seleccionada. Finalmente, examinaremos el estudio de sostenibilidad y el compromiso social asociados al proyecto.

2
Contextualización

En el contexto actual, estamos presenciando un crecimiento exponencial en la generación de datos, una tendencia que se prevé continúe en un futuro inmediato. Específicamente, una parte considerable de estos datos consiste en flujos continuos de información, conocidos como *data streams* o flujos de datos. Estos se generan y transmiten a gran velocidad y volumen, siendo característicos de fuentes como redes sociales, servidores web y mercados financieros. Los *data streams* poseen atributos distintivos: se generan y recolectan continuamente, presentan grandes volúmenes, llegan a alta velocidad con un tiempo limitado para su procesamiento, y tienen un potencial casi ilimitado debido a su generación constante.

Dada la naturaleza compleja y dinámica de los *data streams*, su procesamiento y análisis, conocido como data streaming, requiere enfoques y técnicas especializadas. Estas metodologías varían según los objetivos particulares de cada proyecto. En nuestro caso, nos centramos en la detección de anomalías en el comportamiento de ítems individuales dentro del flujo, tomando en cuenta sus patrones históricos. Esto representa un área de estudio extensa y desafiante, que podría ser demasiado amplia para un Trabajo de Fin de Grado (TFG). Por ello, hemos identificado y destacado las características más relevantes de los algoritmos que deseamos explorar, con el objetivo de definir un alcance más manejable para nuestro TFG.

Antes de sumergirnos en la complejidad de los algoritmos y técnicas para la detección de anomalías, es esencial establecer una base sólida definiendo qué entendemos por *anomalía*. Este concepto será el punto de partida de nuestro análisis y el pilar sobre el que construiremos nuestra investigación y desarrollo. Por tanto, el siguiente subapartado se dedicará a aclarar y profundizar en la definición del concepto de anomalía.

2.1. El concepto de anomalía

La información de este apartado se fundamenta en el paper *A Unifying Review of Deep and Shallow Anomaly Detection* [1]. Este paper propone la siguiente definición de anomalía:

Una anomalía es una observación que se desvía considerablemente de algún concepto de normalidad.

En el ámbito de detección de anomalías, la normalidad se refiere al comportamiento esperado o típico observado en los datos bajo estudio. Esta normalidad puede variar ampliamente dependiendo del contexto y del conjunto de datos específico, y a menudo se establece basándose en patrones históricos y estadísticas descriptivas. Desviación *considerable* sugiere una diferencia significativa con respecto a lo que es típico o esperado, lo cual puede ser identificado a través de métricas estadísticas, distancias o comparaciones con patrones conocidos.

Esta definición, como podemos observar, es bastante genérica. Para entrar en detalle, el artículo clasifica diferentes tipos de anomalías:

- **Anomalía puntual:** Es un punto individual de datos anómalo, como un punto distante del clúster principal en relación con los otros puntos existentes.
- **Anomalía de grupo o colectiva:** Similar al caso anterior, pero ahora se trata de un conjunto de puntos relacionados o dependientes que son anómalos.
- **Anomalía condicional o contextual:** Es una instancia de datos que es anómala en un contexto específico, como el tiempo, el espacio o las conexiones

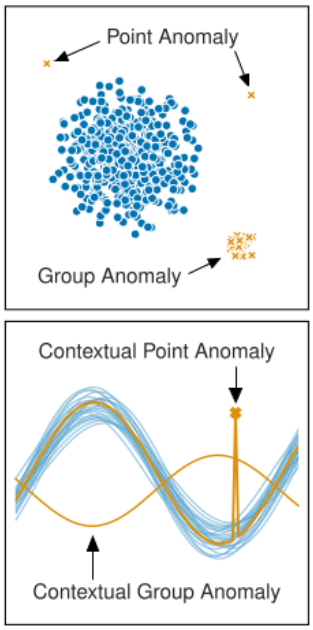

Figura 2.1: Una anomalía puntual es un simple punto anómalo. Una anomalía puntual contextual ocurre cuando un punto se desvía en su contexto local, por ejemplo, un pico en una serie temporal. Una anomalía grupal puede ser un conjunto de anomalías o alguna serie de puntos relacionados que son anómalos bajo la distribución conjunta de la serie (anomalía grupal contextual). Fuente: [1].

en un gráfico. Por ejemplo, un precio de $1 por acción de Apple Inc. podría haber sido normal antes de 1997, pero hoy (2021) sería una anomalía. Es importante tener en cuenta que a menudo las anomalías puntuales y colectivas también son contextuales.

En la figura 2.1 se pueden apreciar claramente estos tipos de anomalías.

Dependiendo del contexto de estudio, se puede hacer una distinción entre anomalía, *outlier* y novedad.

- Una **anomalía** se caracterizan como una instancia de una distribución distinta. Por ejemplo, cuando las anomalías son generadas por un proceso diferente al de los puntos normales.

- Un ***outlier*** es generado con el mismo proceso, pero es una instancia rara o de baja probabilidad.

- Una **novedad** es un punto de datos que no se ajusta a las distribuciones previamente observadas y puede representar un patrón emergente o cambio en el comportamiento de los datos; en otras palabras, es algo nuevo que no se ha visto hasta ahora. Una novedad se suele usar cuando los datos son evolutivos en el tiempo.

Para interpretar adecuadamente cualquiera de estos elementos, es fundamental comprender el contexto en el que emergen. Dependiendo del contexto específico y la naturaleza de los datos, una misma observación puede clasificarse como un *outlier*, una anomalía o una novedad. Profundizaremos en estos tres conceptos durante nuestro estudio del estado del arte. Por el momento, es crucial reconocer que para detectar anomalías de manera efectiva, primero debemos entender qué se considera normal en un conjunto específico de datos. Luego, identificaremos aquellas observaciones que se desvíen significativamente de esa normalidad, considerando los diversos tipos y contextos en los que estas desviaciones pueden manifestarse.

2.2. El carácter exploratorio del proyecto

Antes de entrar a definir los campos de estudio del proyecto, es oportuno hacer énfasis en el carácter exploratorio del proyecto.

Hemos mencionado que queremos realizar un estudio de los algoritmos de detección de anomalías. Dado que al principio no tenemos conocimiento sobre este campo de estudio, es necesario realizar una búsqueda o exploración. En esta fase de exploración, queremos introducirnos en este campo de estudio e intentar encontrar técnicas y algoritmos que puedan ser de ayuda para el problema que hemos propuesto.

Esta fase de exploración, en comparación con la fase de autoaprendizaje de otros trabajos, presenta una diferencia importante: el riesgo. En una fase de autoaprendizaje, el propósito es aprender sobre un conocimiento muy concreto que se sabe que se puede adquirir. En cambio, en nuestra fase de exploración, no sabemos de antemano qué técnicas utilizaremos ni qué hay que aprender. Podemos decir que esta fase de exploración es el paso previo a la fase de autoaprendizaje.

Es importante enfatizar que, debido al carácter exploratorio de este proyecto, algunos materiales o campos de estudio abordados durante esta fase de exploración no se han hecho visibles en el trabajo final. Esto se debe a que, a medida que avanzábamos, identificamos ciertos enfoques y metodologías que, aunque inicialmente parecían prometedores, finalmente se desviaban de los objetivos centrales de nuestro trabajo o, tras un análisis más detallado, resultaron no ser direcciones viables para nuestra investigación.

Este proceso de filtrado y selección es una parte inherente de la naturaleza exploratoria del proyecto. No todas las vías exploradas conducen a resultados que se alineen directamente con nuestros objetivos, pero cada una de ellas ha contribuido a la comprensión más amplia del campo y ha informado nuestras decisiones finales.

Una vez definido el carácter exploratorio del proyecto, lo siguiente es saber encontrar una dirección. Para esto, necesitamos realizar un análisis más detallado de nuestros objetivos.

2.3. Análisis del objeto de estudio

El objetivo de estudio son los *data streams*. Esto implica que tenemos restricciones en el procesamiento del *data stream*:

- **Restricción temporal:** estamos en un contexto de un gran flujo de datos. La velocidad de recepción es muy rápida, por lo que el procesamiento también tiene que ser rápido. Esto nos limita, por ejemplo, a acceder a los datos en el orden de recepción.

- **Restricción espacial:** nos llega un gran volumen de datos, y no tenemos suficiente espacio para guardarlos todos. En el procesamiento del *data stream*, necesitamos un mecanismo que elimine datos antiguos para aliviar espacio y poder almacenar nuevos datos.

En apartados anteriores, hemos definido los diferentes tipos de anomalías y los términos similares a anomalía: *outlier* y novedad. Para realizar un análisis más detallado, consideramos necesario crear un modelo de datos.

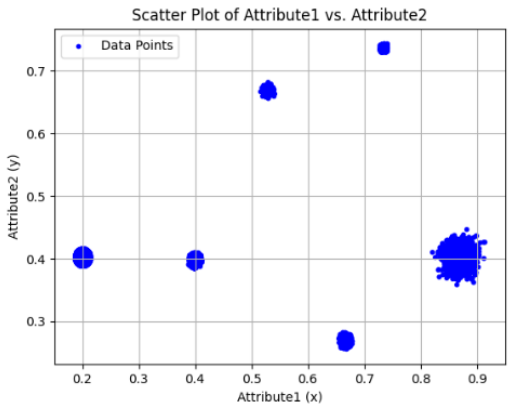

Figura 2.2: Clústeres compuestos por datos bidimensionales. Generado con el generador de clústeres mdcgenpy [2].

Definiremos las características del *data stream* que trataremos. Esto nos ayudará a reducir el dominio de estudio.

Dado un *data stream*, cada entrada del *stream* la llamamos ítem. A cada ítem le corresponde una tupla, en la que cada componente representa un atributo. Por ejemplo, si los ítems son puntos en un espacio bidimensional, entonces sus atributos serán una tupla de números reales (x, y). Además, podemos agrupar estos ítems en conjuntos según el valor de sus atributos. Siguiendo el ejemplo anterior, podemos agrupar los puntos en clústeres según el valor de sus coordenadas x e y, como podemos ver en la figura 2.2.

Partiendo de este modelo, podemos realizar pequeñas modificaciones según la anomalía, el *outlier* o la novedad que queramos estudiar (dependiendo de lo que encontremos). Para cada caso, posibles modificaciones podrían ser:

- **Anomalía:** Generar puntos que estén distantes de todos los clústeres existentes, dificultando su clasificación. Este sería un caso de anomalía contextual puntual.

- ***outlier*:** En este caso, no tenemos que realizar ninguna modificación. Simplemente generando los clústeres siguiendo cierta distribución de probabilidad, ya podemos obtener *outliers*.

- **Novedad:** Aunque al principio no lo consideramos como objeto de estudio, hemos encontrado un campo de investigación que busca objetivos muy similares a los que buscamos. En este caso, el objeto de estudio son las novedades. En el contexto del *data streaming*, una novedad es un elemento nuevo difícil de clasificar con el modelo actual. Por lo tanto, se buscará actualizar dicho modelo para que pueda clasificar el nuevo elemento y futuros elementos similares a este.

2.4. Actores implicados

En este proyecto, diversos grupos e individuos mantienen un interés directo o indirecto en su evolución y resultados. El resultado de este proyecto se compartirá públicamente, probablemente a través de una plataforma como GitHub. A continuación, se identifican los actores implicados, resaltando su relación e interés en el proyecto:

1. **Estudiantes e investigadores en ciencias de la computación y datos**: Este grupo incluye a los estudiantes e investigadores interesados en la detección de anomalías u otros temas relacionados. Buscan aplicar y profundizar en los conocimientos y herramientas desarrolladas, esperando que estos recursos fortalezcan sus propios proyectos e investigaciones.

2. **Participantes en el desarrollo del proyecto**

    - **Estudiante responsable del proyecto**: Representado por He Chen. Como actor principal, ha sido el responsable de la conceptualización, desarrollo e implementación del proyecto. Sus tareas han incluido la investigación del estado del arte, selección y estudio de los algoritmos apropiados, implementación del generador de datos sintéticos, adaptación y mejora del código de SENCForest, así como la realización de experimentos y análisis de resultados.
    - **Director del Proyecto**: Representado por Conrado Martínez Parra. Como actor secundario, ha proporcionado supervisión, orientación y

apoyo académico a lo largo del desarrollo del proyecto. Ha contribuido con su experiencia en el campo, asesorando en la selección de métodos, proporcionando recursos y facilitando el acceso a literatura relevante. Su rol ha sido fundamental en la dirección y refinamiento del enfoque investigativo del proyecto.

3. **Facultad de Informática de Barcelona y Universidad Politécnica de Cataluña**

   Como instituciones educativas involucradas en el proyecto, buscan fomentar la investigación y el desarrollo académico. Están interesadas en que el proyecto refleje la calidad educativa y de investigación de la universidad, contribuyendo a su prestigio y al avance del conocimiento.

4. **Comunidad de Desarrolladores de Software**

   Este grupo se beneficia indirectamente del proyecto, ya que pueden utilizar las herramientas y algoritmos desarrollados en el TFG para sus propios fines. Los desarrolladores de software, tanto profesionales como aficionados, pueden aplicar estas herramientas en sus proyectos, extendiendo así el impacto y la aplicabilidad del trabajo realizado.

5. **Usuarios del Software Desarrollado**

   Este grupo incluye a los individuos y organizaciones que utilizan el software desarrollado mediante nuestras herramientas, que se beneficia de algoritmos de detección de anomalías más fiables, robustos y eficientes gracias a las lecciones aprendidas.

Es importante enfatizar que este proyecto, de naturaleza puramente académica, tiene el potencial de servir como referencia para otros estudiantes e investigadores. Esto se debe a que todos los materiales y resultados de este trabajo serán publicados en GitHub [3], facilitando el acceso y la consulta. En el mejor de los casos, este proyecto podría incluso despertar el interés de estudiantes que busquen continuar o expandir esta línea de investigación.

3
Alcance del proyecto

3.1. Objetivos

El objetivo principal de este proyecto es llevar a cabo un estudio detallado algoritmos y estructuras de datos utilizados para la detección de anomalías en *data streams*. Estos algoritmos y estructuras de datos se aplicarán para abordar los desafíos previamente mencionados en las secciones anteriores del proyecto. Además, se busca disponer de estos algoritmos en forma de software listo para su utilización, lo que podría lograrse mediante la creación de una biblioteca de Python u otra forma equivalente. Es importante destacar que el proyecto no implica la creación de nuevos algoritmos, sino la exploración y utilización de herramientas existentes en el contexto del proyecto.

Para lograr estos objetivos, se debe construir un entorno de software apropiado que incluya las siguientes etapas:

1. **Implementación de generadores de datos sintéticos**

   Dado que la adquisición de datos útiles para la experimentación con algoritmos de detección de anomalías en *data streams* puede ser un desafío, se propone la creación de generadores sintéticos de datos que modelen datos reales. Estos generadores deben ser capaces de generar *data streams* con características

específicas, y también deben proporcionar información sobre la ubicación real de las anomalías en los datos generados.

2. **Definición de estándares de medida**

   Es esencial establecer métricas estándar que permitan evaluar el rendimiento de los algoritmos y comparar su eficacia en diferentes conjuntos de datos.

3. **Experimentación**

   Una vez que se tenga el entorno adecuado y los algoritmos estén preparados, se procederá a la fase de experimentación. En esta etapa, se analizará el comportamiento de los algoritmos utilizando diferentes conjuntos de datos generados por los generadores sintéticos previamente desarrollados.

El alcance de este proyecto se centra en investigar algoritmos y técnicas relacionados con la detección de anomalías en *data streams*, así como en proporcionar una implementación práctica y un entorno de experimentación para llevar a cabo este estudio de manera efectiva.

3.2. Riesgos

En general, se pueden distinguir tres fases en el proyecto: la fase de exploración, la fase de aprendizaje y la fase de desarrollo. Los principales riesgos están asociados con la fase de exploración, cuyos impactos serán importantes para la planificación que se detallará en el siguiente capítulo.

Durante la fase de exploración de un proyecto de detección de anomalías, se enfrentan riesgos específicos vinculados con la naturaleza investigativa y el terreno frecuentemente inexplorado de esta fase. A continuación, se exponen los riesgos centrados en esta etapa y las implicaciones que pueden conllevar:

1. **Incertidumbre**:

- **Riesgo**: El inicio de la exploración trae consigo una considerable incertidumbre tanto en aspectos técnicos como conceptuales. Sin una base de conocimiento sólida y definida, elegir la dirección adecuada para la investigación y el desarrollo puede ser un reto.
- **Implicación**: Es posible que se inviertan recursos significativos en explorar rutas que finalmente resulten infructuosas o menos eficientes, lo cual podría retrasar el avance del proyecto.

2. **Subestimación de la complejidad**:

   - **Riesgo**: La complejidad de los algoritmos y técnicas en el ámbito de la detección de anomalías puede ser mayor de lo esperado, requiriendo una comprensión profunda y habilidades especializadas.
   - **Implicación**: Podría subestimarse el tiempo y los recursos necesarios para adquirir el conocimiento y la competencia requeridos, llevando a retrasos y posibles compromisos en la calidad del proyecto.

3. **Cambio en los objetivos del proyecto**:

   - **Riesgo**: Nuevos descubrimientos durante la fase exploratoria pueden exigir una reevaluación y ajuste de los objetivos y estrategias iniciales del proyecto.
   - **Implicación**: Modificar los objetivos puede provocar una serie de ajustes en la planificación y el enfoque del proyecto, impactando en la gestión del tiempo y los recursos.

4. **Dificultades en la selección de técnicas**:

   - **Riesgo**: La tarea de identificar y seleccionar entre una amplia gama de técnicas y herramientas disponibles puede ser abrumadora y conducir a elecciones subóptimas.

- **Implicación**: Optar por una técnica inadecuada no solo puede retrasar el proyecto, sino también comprometer la validez y eficacia de los resultados.

Para mitigar estos riesgos, es crucial adoptar un enfoque flexible e informado. Esto implica dedicar tiempo suficiente a la revisión de literatura, consultar con expertos en el campo, establecer hitos claros para la reevaluación constante de la dirección del proyecto y mantener una comunicación abierta y regular con el director del proyecto. Además, es esencial estar preparado para ajustar los objetivos y métodos conforme se adquiere un mayor conocimiento y comprensión del campo.

3.3. Posibles obstáculos

Además de los riesgos en la fase de exploración, es importante destacar los posibles obstáculos en la fase de desarrollo una vez que tengamos clara la dirección de la investigación. Aunque estos obstáculos no son tan impactantes a nivel de los riesgos que pueden afectar la planificación, también deben considerarse para asegurar la calidad de los resultados.

1. **Calidad de datos y variabilidad**: La detección de anomalías puede ser complicada si las anomalías son sutiles o no están claramente representadas en los datos. Problemas como la calidad de los datos, valores faltantes y la variabilidad en los flujos de datos pueden dificultar la identificación precisa de anomalías.

2. **Escalabilidad**: Manejar flujos de datos de alta velocidad con un gran número de elementos puede ser intensivo en recursos. Es importante asegurar que el entorno de software sea escalable y capaz de manejar eficientemente un aumento en el volumen de datos para garantizar su efectividad en entornos de producción.

3. **Falsos positivos y negativos**: Encontrar el equilibrio adecuado entre detectar anomalías (minimizar falsos negativos) y evitar alarmas falsas (minimizar

falsos positivos) es un desafío común en la detección de anomalías. Optimizar este equilibrio puede requerir ajustes y refinamientos en los algoritmos.

4. **Métricas de evaluación**: Definir métricas de evaluación adecuadas para el sistema de detección de anomalías puede ser complicado. Es crucial decidir cómo medir el rendimiento de manera precisa, especialmente cuando se trabaja con conjuntos de datos desequilibrados. La elección de métricas adecuadas es esencial para una evaluación precisa de los algoritmos.

Al reconocer y abordar estos obstáculos, podemos mejorar la robustez y la fiabilidad del sistema desarrollado en la fase de desarrollo, contribuyendo así a la calidad general del proyecto.

3.4. Metodología y rigor

El proyecto se llevará a cabo siguiendo una metodología estructurada que consta de varias fases principales:

1. **Fase de exploración**: Esta etapa inicial es fundamental para comprender el terreno y las opciones disponibles. Implica una inmersión profunda en el estado del arte para explorar y entender las diferentes técnicas, algoritmos y herramientas disponibles en el campo de la detección de anomalías. La comunicación constante con el director del proyecto es vital en esta fase para guiar la exploración, discutir hallazgos y ajustar la dirección del proyecto según sea necesario. El resultado de esta fase será la selección de una dirección específica o conjunto de técnicas que se investigarán más a fondo.

2. **Aprendizaje autónomo para especializarse en el tema**: Basándose en la dirección escogida en la fase de exploración, el estudiante dedicará tiempo a estudiar de manera autónoma el tema específico de detección de anomalías en *data streams* relacionado con la dirección seleccionada. Esto incluirá la revisión de literatura relevante, la investigación profunda de los algoritmos

y técnicas seleccionadas y la comprensión avanzada de los conceptos clave relacionados con el área de enfoque.

3. **Implementación del entorno de software para los experimentos**: Con un conocimiento más profundo y especializado adquirido, se procederá a la implementación del entorno de software necesario para llevar a cabo los experimentos. Esto incluirá la creación de generadores sintéticos de datos y la configuración de métricas de evaluación.

4. **Adaptación del algoritmo seleccionado al generador de datos:** Esta etapa implica comprender y adaptar el algoritmo seleccionado para su uso con el generador de datos desarrollado. Como hemos mencionado anteriormente, el enfoque del proyecto no es crear un nuevo algoritmo, sino explorar y analizar algoritmos ya existentes. Una consideración importante en esta fase es el lenguaje de programación en el que está escrito el algoritmo seleccionado. Si no se domina este lenguaje, se incurre en un coste adicional de aprendizaje y tiempo para comprender cómo utilizar el algoritmo en nuestros experimentos. Además, se debe considerar el esfuerzo requerido para adaptar los datos generados por nuestro generador a las especificaciones y requerimientos del algoritmo seleccionado.

5. **Experimentación**

   Una vez que se hayan implementado los algoritmos y estructuras de datos, se llevará a cabo una serie de experimentos. Estos experimentos involucrarán la evaluación del rendimiento de los algoritmos en diferentes conjuntos de datos generados por los generadores sintéticos. Se recopilarán datos y resultados para su análisis.

Durante todas estas fases del proyecto, se mantendrá una comunicación constante entre el estudiante y el director del proyecto a través de la mensajería en Slack. La planificación de tareas y el seguimiento de actividades se gestionarán mediante

Trello [4], lo que permitirá mantener un registro claro de las actividades realizadas y las pendientes.

Además, la compartición de material, como código y documentación, se llevará a cabo mediante plataformas como Github y Overleaf [5], lo que facilitará la colaboración y la revisión conjunta.

Se programarán reuniones semanales en el Edificio Omega del Campus Nord con el objetivo de realizar un seguimiento continuo del trabajo, aclarar dudas y planificar las próximas tareas. Estas reuniones presenciales permitirán una interacción directa y efectiva entre el estudiante y el director del proyecto.

Esta metodología estructurada y los procedimientos de comunicación y seguimiento establecidos garantizarán un enfoque riguroso y efectivo en la ejecución del proyecto.

4
Planificación temporal

Comenzamos estimando la duración total del proyecto. Con la ayuda del calendario académico, podemos determinar que el período del proyecto comprende desde la semana del 12 de septiembre de 2023 hasta la semana del 22 de enero de 2024, que es cuando comienza el período de presentación del TFG. En total, hay 19 semanas comprendidas en estas dos fechas (sin contar la semana de presentación del TFG). Estimando que dedicaremos aproximadamente 25 horas por semana al proyecto, la duración estimada del proyecto sería de unas 475 horas en total.

4.1. Recursos necesarios

A lo largo del proyecto, se necesitarán cuatro tipos de recursos:

- Recursos humanos [R1]: Incluyen al director del proyecto, el tutor de GEP y el programador(estudiante).
- Fuentes de investigación [R2]: Para conocer los algoritmos existentes relacionados con el tema, se requerirá acceder a informes de investigación (*research papers*). Es posible que para algunos de estos informes se necesiten permisos o sea necesario pagar por su acceso. En esta situación, se recurrirá al director del proyecto, quien tiene más recursos en este campo. No se considerarán informes de investigación que no estén disponibles de forma gratuita.

- Recursos de hardware [R3]: Se necesitará una computadora con conexión al Internet para llevar a cabo las tareas de programación, en este caso es un portátil personal del estudiante.

- Recursos de software [R4]: Solo se utilizarán programas gratuitos o aquellos que estén disponibles gratuitamente para estudiantes. Se utilizarán GitHub (control de versiones), Trello (organización de tareas), PyCharm (IDE de Python), Matlab y Overleaf (documentación en LaTeX).

4.2. Descripción de tareas

El proyecto se puede dividir en cinco bloques según su finalidad: gestión de proyecto, estudio previo, desarrollo de los programas y experimentos. Puedes encontrar el resumen de las tareas en la tabla 4.1. También lo puede encontrar en forma de diagrama de Gantt 4.1.

4.2.1. Gestión de proyectos

Esta sección abarca tareas relacionadas con la asignatura de GEP, reuniones de seguimiento y la elaboración de la memoria del proyecto.

Contexto y alcance

En esta parte del proyecto, se establece el contexto, el alcance, los objetivos y la metodología. Es esencial para definir claramente qué se construirá y por qué, proporcionando una base sólida para el trabajo.

Planificación temporal

Aquí se describen las tareas necesarias para llevar a cabo el proyecto, estimando el tiempo requerido para cada una y su secuencia de ejecución.

Presupuesto y sostenibilidad

Esta sección incluye una estimación del costo económico del proyecto, en función de los recursos utilizados. También se realizará un análisis de los efectos en términos de sostenibilidad.

Informe de seguimiento

Es un documento que recoge los cambios o mejoras en los objetivos y en la planificación del proyecto.

Memoria

La memoria es un informe escrito que presenta y documenta la investigación realizada, los resultados obtenidos y las conclusiones derivadas del proyecto. Es una tarea que se llevará a cabo durante todo el proyecto.

Presentación

Aquí se incluyen los trabajos necesarios para realizar la defensa del proyecto, que abarcan la preparación de los materiales de presentación, la pauta de presentación y los ensayos.

Reuniones

Se refiere a las reuniones con el director del proyecto, donde se realiza un seguimiento del mismo. Se ha acordado que estas reuniones serán semanales.

4.2.2. Aprendizaje

Abarca todas las actividades relacionadas con la preparación y comprensión de los conceptos, herramientas y conocimientos necesarios para llevar a cabo el proyecto.

Exploración

Son todas las actividades en la fase de exploración. Incluyendo lectura y comprensión de artículos de investigación, se llevará a cabo a través de la revisión de los artículos de investigación, análisis de los algoritmos y técnicas, y la decisión

de una dirección de investigación. No buscamos profundizar al detalle sobre un determinado algoritmo o técnica, sino determinar en qué ámbito u ámbitos concretos debemos centrarnos. Para ello, necesitamos analizar diferentes técnicas existentes y seleccionar las más apropiadas para el estudio que queremos realizar.

Estudio de algoritmos

Una vez hemos adquirido suficiente conocimiento sobre el tema y hemos decidido el ámbito de estudio, podemos entrar a hacer un estudio más detallado de algoritmos o técnicas más concretas.

Herramientas para el desarrollo

Para desarrollar la solución escogida hace falta el uso del lenguaje Python con algunas de sus librerías, por ejemplo, Numpy, Scikit-learn, Pandas, que ofrecen una gran eficiencia para el análisis de datos. Otra herramienta que se usará es Matlab. Se ha hecho un repaso de cómo usar estas dos herramientas.

4.2.3. Desarrollo

El bloque de desarrollo es una etapa donde se concretan las ideas y se materializan las soluciones. Está compuesto por tres subtareas esenciales: implementación del generador de datos, adaptación y aprendizaje de los códigos de algoritmos para nuestros experimentos, y la ejecución de los experimentos propiamente dichos.

Implementación del generador de datos

Esta tarea se enfoca en la creación de un generador de datos sintéticos que simule los flujos de datos en los que se buscarán anomalías. El generador debe ser capaz de producir datos con características y comportamientos variados, incluyendo instancias anómalas, para probar la efectividad de los algoritmos de detección.

Codigo	Tarea	Duración	Dependència	Recursos
T1	**Gestión del projecto**	**160 h**		
T1.1	Contexto y alcance	15 h		R1, R3
T1.2	Planificación temporal	10 h	T1.1	R1, R3
T1.3	Presupuesto y sostenibilidad	10 h	T1.1, T1.2	R1, R3
T1.4	Informe de seguimiento	20		R1, R3
T1.5	Memoria	80 h		R1, R3
T1.6	Presentación	10 h	T1.4	R1, R3
T1.7	Reuniones	15 h		R1, R3
T2	**Aprendizaje**	**160h**		
T2.1	Exploración	100 h		R1, R2
T2.2	Estudio de algoritmos	50 h		R1, R2
T2.3	Herramientas para el desarrollo	10h		R1, R3, R4
T3	**Desarrollo**	**155 h**	**T2**	
T3.1	Generador de Datos	85 h		R1, R3, R4
T3.2	Adaptar código de los algoritmos	40 h		R1, R3, R4
T3.3	Experimentos	30 h	T3.1, T3.2	R1, R3, R4

Cuadro 4.1: Resumen de las tareas

Aprender y adaptar los códigos del algoritmo

Esta subtarea se centra en entender y utilizar algoritmos escogidos ya existentes y listos para usar. Requiere estudiar cómo funcionan estos algoritmos, aprender a operarlos con los datos proporcionados por nuestro generador y adaptar cualquier parámetro o configuración para optimizar su rendimiento en nuestros experimentos específicos.

Experimentos

Se llevarán a cabo pruebas para evaluar el rendimiento y la eficacia de los algoritmos. Estas pruebas incluirán la generación de datos de prueba, la ejecución de los algoritmos en estos datos y el análisis de los resultados obtenidos.

4.3. Gestión de riesgo

La gestión de riesgos es un componente crucial en la planificación de cualquier proyecto, especialmente aquellos de carácter investigativo y técnico como el presente. Como se detalló en la sección de riesgos (Sección3.2), hemos identificado cuatro

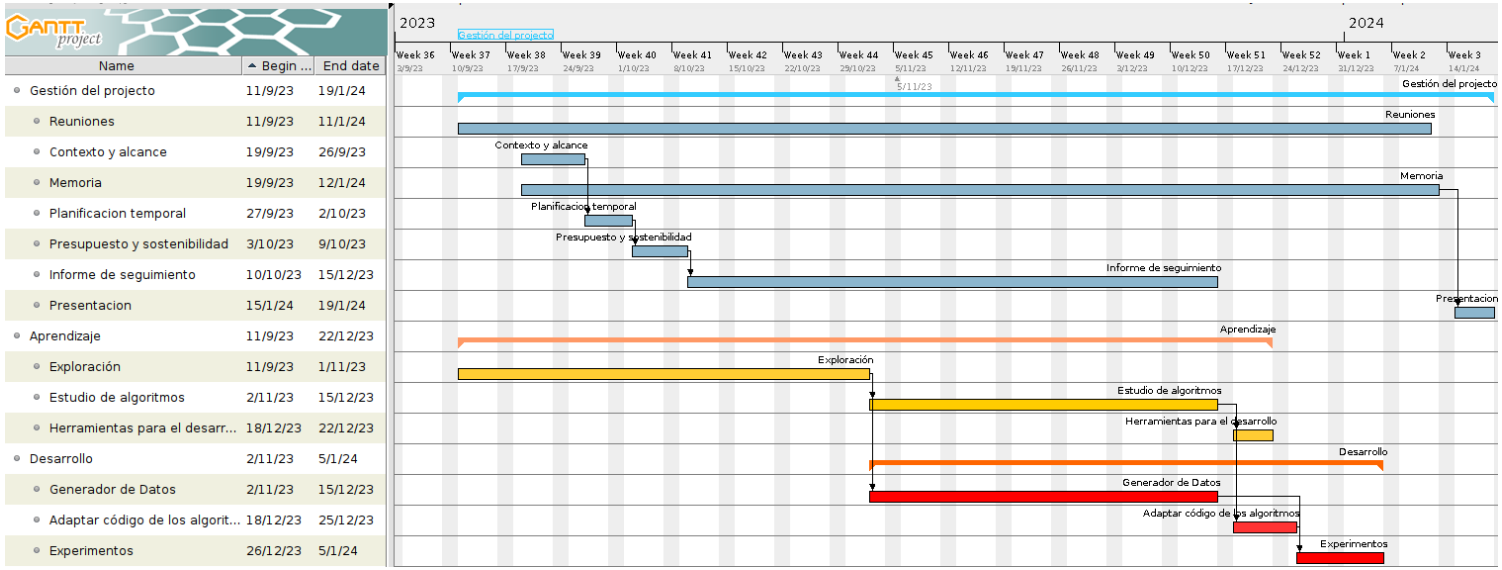

Figura 4.1: Diagrama de gantt.

riesgos potenciales: Incertidumbre, Subestimación de la Complejidad, Cambio en los Objetivos del Proyecto y Dificultades en la Selección de Técnicas. Para mitigar estos riesgos, adoptaremos las siguientes estrategias:

1. **Subestimación de tiempo y recursos**: Realizaremos una evaluación conservadora del tiempo y los recursos necesarios para las tareas de aprendizaje, enfocándonos específicamente en la Exploración y el Estudio de Algoritmos. El impacto de esta medida se reflejará en un aumento de las horas previstas para estas dos tareas en la planificación del proyecto.

2. **Manejo de la incertidumbre**: Para abordar la incertidumbre al comienzo de la fase de exploración, implementaremos sesiones regulares de brainstorming y consultas con el director del proyecto u otros expertos en el campo. Además, se establecerán evaluaciones periódicas para revisar el progreso y entender la complejidad a medida que avanza el proyecto. También se organizarán sesiones de capacitación para asegurar que el equipo adquiera las habilidades necesarias para enfrentar las complejidades técnicas. Esta estrategia incluirá reuniones periódicas con el director del proyecto y el mantenimiento de una comunicación constante y efectiva con él.

Estas medidas están diseñadas para proporcionar un marco flexible y adaptativo que permita al proyecto navegar eficientemente a través de las incertidumbres y desafíos de la investigación, garantizando al mismo tiempo que los recursos se utilicen de manera eficiente y que los objetivos del proyecto se mantengan alineados con las realidades y descubrimientos emergentes.

5 Gestión económica

En esta sección, se estimarán los costos necesarios para el desarrollo exitoso del proyecto. Además, se elaborará un plan de contingencia para abordar posibles desviaciones en el presupuesto.

5.1. Presupuesto

5.1.1. Costos de personal

El personal involucrado en este proyecto incluye:

- Director del proyecto (profesorado): Participa en la planificación inicial del proyecto, contribuye al diseño general del programa y está involucrado en la etapa de investigación.

- Programador (estudiante): Principalmente responsable del desarrollo del programa y también participa en la investigación.

En la tabla 5.1 recoge el coste por hora de los personales y en la tabla 5.2 recoge el coste por tarea de los personales.

ID	**Rol**	**Coste por hora**
D	Director del proyecto	25 euros/h [6]
P	Programador (estudiante de la FIB)	10 euros/h [7]

Cuadro 5.1: Costes de personal por hora

Codigo	**Tarea**	**Duración**	**Roles**	**Coste (euros)**
T1	**Gestión del projecto**	**115 h**		2502.5
T1.1	Contexto y alcance	15 h	D	487.5
T1.2	Planificación temporal	10 h	D	325
T1.3	Presupuesto y sostenibilidad	10 h	D	325
T1.4	Memoria	60 h	P	780
T1.5	Presentación	10 h	P	130
T1.6	Reuniones	10 h	D, P	455
T2	**Aprendizaje**	**60h**		
T2.1	Python y sus herramientas	10 h	P	130
T2.2	Estudio de algoritmos	50 h	D, P	1690
T3	**Desarrollo del prototipo**	**130 h**		
T3.1	Diseño de la arquitectura	10 h	P	130
T3.2	Implementación	110 h	P	1430
T3.3	Validación	10 h	P	130
T4	**Desarrollo del producto final**	**120 h**		1560
T4.1	Diseño de la arquitectura	20 h	P	260
T4.2	Implementación	90 h	P	1170
T4.3	Validación	10 h	P	130
T5	**Experimentos**	**50 h**	P	650
	Coste Total			**4,570**

Cuadro 5.2: Coste de personal por tarea. Coste = Duración* Coste por hora * 1.3, donde 1.3 es la parte de la seguridad social

5.1.2. Costos de software

Se utilizarán GitHub (control de versiones), Trello (organización de tareas), PyCharm (IDE de Python) y Overleaf (documentación en LaTeX). Para calcular los costos de software, primero necesitamos identificar los costos asociados a cada herramienta y luego sumarlos. Dado que PyCharm es el único software de pago, calcularemos su costo total.

- Costo mensual de PyCharm = 30.13 euros
- Número de meses de uso = 5 meses

- Costo total de PyCharm = 30.13 euros/mes * 5 meses = 150.65 euros

Por lo tanto, el costo total de PyCharm durante 5 meses es de 150.65 euros.

Sin embargo, dado que el programador es un estudiante, puede utilizar PyCharm de forma gratuita. Los otros software se pueden usar de forma gratuita. Al final, resulta que no tenemos costos de software.

5.1.3. Costos de hardware

Para la realización del proyecto, solo se necesita un ordenador personal para el programador, que se utilizará para el desarrollo del proyecto. Concretamente, se utilizará un portátil del modelo Mi Laptop Pro 15.6, que tiene un precio de unos 1200 euros. La vida útil estimada de este portátil es de unos 4 años. Debido a que este ordenador se utilizará para casi todas las tareas definidas en el proyecto, que suman un total de 475 horas, podemos calcular el costo de hardware de la siguiente manera:

1 año = 365 días = 24 horas por día = 8,760 horas por año.

Entonces, la vida útil del portátil en horas sería: 4 años * 8,760 horas por año = 35,040 horas.

Ahora, podemos calcular el costo de hardware nuevamente: Costo de hardware = (Precio del portátil / Vida útil del portátil en horas) * Horas de utilización estimadas.

Costo de hardware = (1200 euros / 35,040 horas) * 475 horas = 16.13 euros.

Por lo tanto, el costo de hardware para el proyecto se estima en aproximadamente 16.13 euros.

5.1.4. Contingencia

En cualquier proyecto, es esencial tener en cuenta la posibilidad de enfrentar desafíos imprevistos o problemas que pueden surgir durante el desarrollo. Al tratarse de un proyecto de investigación existe una alta probabilidad de que surjan dificultades inesperadas. Por lo tanto, para asegurarnos de que estamos preparados para abordar estos problemas, hemos decidido agregar un margen adicional del 30 % al presupuesto original del proyecto. Esto nos proporcionará recursos adicionales para manejar

Tipo de coste	Coste	Contingencia
Costos de Personal	4570	1371
Costos de Software	0	0
Costos de Hardware	16.13	4.839
Total	**4586.13**	**1321.839**

Cuadro 5.3: Contingencia del 30 % por tipo de coste

cualquier contratiempo que pueda surgir y garantizar que podamos completar el proyecto de manera exitosa y sin interrupciones importantes.

En la tabla 5.3 se detalla la contingencia del proyecto:

5.1.5. Imprevistos

Hay que tener en cuenta una serie de imprevistos con mayor probabilidad de suceder. En la tabla 5.4 se encuentra un resumen de todos los sobrecostes.

- **Aumento del tiempo de aprendizaje**: Dado el carácter investigador del proyecto, existe una considerable incertidumbre en la tarea de estudio de algoritmos. Para abordar esta incertidumbre, hemos decidido agregar un margen de seguridad del 50 %, lo que equivale a 25 horas adicionales de aprendizaje. Esto representa un costo adicional de 845 euros en términos de recursos humanos. Consideramos que existe un riesgo del 30 % de que se requiera este tiempo adicional debido a la complejidad de los algoritmos y la investigación necesaria.

- **Aumento del tiempo de desarrollo**: Dado que el programa que deseamos desarrollar debe tener una estructura modular y puede presentar dificultades técnicas imprevistas, hemos decidido incluir un margen de 30 horas adicionales de desarrollo del producto para el programador. Esto implica un costo adicional de 390 euros en recursos humanos. Estimamos que existe un riesgo del 20 % de que se necesite este tiempo adicional debido a desafíos técnicos inesperados.

- **Fallo de hardware**: El proyecto se llevará a cabo utilizando el ordenador personal del programador, por lo que es importante considerar el riesgo

Imprevisto	Coste (euros)	Probabilidad	Coste total (euros)
Aumento del tiempo de aprendizaje	845	30 %	253.5
Aumento del tiempo de desarrollo	390	20 %	78
Fallo de Portátil	1200	5 %	60
Total			**391.5**

Cuadro 5.4: Sobrecoste añadido por imprevistos

Tipo	Coste (euros)
Personal	4570
Software	0
Hardware	16.13
Contingencia	1321.84
Imprevistos	391.5
Total	6299.47

Cuadro 5.5: Presupuesto final del proyecto

de un posible fallo de la máquina. Dado que el precio del portátil es de aproximadamente 1200 euros y estimamos que existe una probabilidad del 5 % de que ocurra un fallo.

5.1.6. Coste total

Teniendo todos los costes calculados, podemos determinar el coste total del proyecto. En la tabla 5.5 se detallan los cálculos:

5.2. Control de gestión

Se llevarán a cabo controles de gestión en cada reunión semanal del proyecto. Además, se realizará una revisión al finalizar cada una de las etapas principales del proyecto.

5.2.1. Proceso de control

- **Reuniones semanales de seguimiento**: Durante las reuniones semanales, se evaluará el progreso del proyecto y se actualizará el presupuesto y los gastos adicionales. Se discutirán las posibles desviaciones y se tomarán medidas correctivas según sea necesario.

- **Revisión al finalizar etapas**: Al finalizar cada etapa importante del proyecto, se realizará una revisión detallada del presupuesto y los gastos. Esto permitirá detectar cualquier desviación significativa y tomar medidas si es necesario.

5.2.2. Registro de información

Se mantendrá un registro detallado de las horas invertidas en cada tarea, los gastos adicionales y cualquier cambio en el presupuesto. Este registro estará disponible para consulta en cualquier momento.

5.2.3. Acciones correctivas

Si se detectan desviaciones en el presupuesto o los gastos, se tomarán medidas correctivas de inmediato. Estas acciones pueden incluir:

- **Ajuste del presupuesto**: Si se superan los límites presupuestarios, se analizarán las posibles áreas donde se puedan realizar ajustes y se tomarán decisiones para mantener el control de los costos.
- **Priorización de tareas**: En caso de desviaciones significativas, se puede considerar la reevaluación de las prioridades y la posible reorganización de las tareas para asegurar que se cumplan los objetivos más críticos.

6
Requisitos de los algoritmos

Para abordar eficazmente la detección de anomalías, *outliers* o novedades en el contexto de *data streams*, es esencial que los algoritmos seleccionados cumplan con ciertos requisitos críticos. Estos requisitos no solo garantizan la precisión y eficiencia de la detección, sino que también aseguran que el algoritmo sea viable y efectivo en un entorno de flujo de datos en tiempo real. A continuación, se presentan y justifican los requisitos clave:

1. **Objetivo de detección de anomalías**:

   - Requisito esencial: El algoritmo debe estar específicamente diseñado para detectar anomalías, *outliers* o novedades en *data streams*.

2. **Eficiencia en tiempo real**:

   - Requisito: Capacidad para procesar y analizar datos en tiempo real o cercano a él. El análisis de datos en tiempo real conlleva:

     a) Los datos tienen que ser procesados en el orden de llegada.

     b) Los datos tienen que ser procesados al momento de llegada.

- Justificación: Los *data streams* se caracterizan por su naturaleza rápida y continua. La habilidad de procesar y hacer predicciones de manera inmediata es esencial para aplicaciones críticas donde la toma de decisiones oportuna basada en los datos más recientes puede ser crucial.

3. **Eficiencia espacial**:

   - Requisito: Mecanismos para eliminar datos y patrones obsoletos, liberando memoria para nuevas instancias.
   - Justificación: Dado el volumen potencialmente infinito de *data streams*, no es práctico ni posible almacenar todos los datos. Un algoritmo eficiente debe poder descartar información antigua o irrelevante para mantener la relevancia y eficiencia del sistema.

4. **Adaptabilidad**:

   - Requisito: Capacidad de actualizar modelos adaptativamente con nuevos datos, sin requerir un entrenamiento completo desde el principio.
   - Justificación: Los patrones en los *data streams* pueden cambiar con el tiempo (fenómeno conocido como *concept drift*). Un algoritmo adaptable puede ajustarse a estos cambios, manteniendo su precisión y relevancia sin la necesidad de reentrenamiento completo, lo cual sería computacionalmente costoso.

5. **Manejo de datos no etiquetados**:

   - Requisito: Funcionamiento eficaz en entornos donde los datos de entrenamiento etiquetados son escasos o inexistentes.
   - Justificación: En muchas aplicaciones de *data streams*, obtener etiquetas es costoso, lento o imposible. Los algoritmos no supervisados, que pueden trabajar con conjuntos de datos de entrenamiento no etiquetados, son particularmente valiosos, ya que pueden operar en un mayor rango de contextos y proporcionar conocimiento valioso.

El primer requisito es indispensable, ya que define el objetivo principal del proyecto. Los demás requisitos serán utilizados para evaluar la eficacia de los algoritmos candidatos.

7
Análisis de alternativas

En esta sección, proporcionaremos un resumen conciso de los candidatos seleccionados tras la fase exploratoria. No profundizaremos en detalles específicos de cada uno, sino que evaluaremos su desempeño en función de sus características y funcionalidades. Utilizaremos los requisitos detallados en el apartado anterior para realizar este análisis. Dada la complejidad de los algoritmos considerados en este proyecto y el tiempo limitado para un estudio de cada uno, es esencial proceder a una selección cuidadosa. Esta elección de candidatos se basará en una evaluación objetiva según los criterios establecidos; sin embargo, es importante destacar que, además de los requisitos técnicos, el interés y la orientación de los investigadores juegan un papel decisivo en este proceso.

En un principio, comenzamos explorando un campo muy estudiado: la detección de *outliers*. En este contexto, se ha utilizado el término *outliers*, aunque en muchos algoritmos de este campo no hacen diferencia entre anomalías y *outliers*; simplemente intentan encontrar elementos que (1) son diferentes de la norma con respecto a sus características; (2) son poco comunes en un conjunto de datos en comparación con las instancias normales. Algunos algoritmos conocidos para la detección de *outliers* incluyen *kNN*, *Isolation Forest* y *k-means*, entre otros. El propósito no es analizar estos algoritmos en detalle, sino encontrar alguna pista de hacia dónde avanzar. Por ejemplo, el artículo [8] explica las dificultades en la detección de

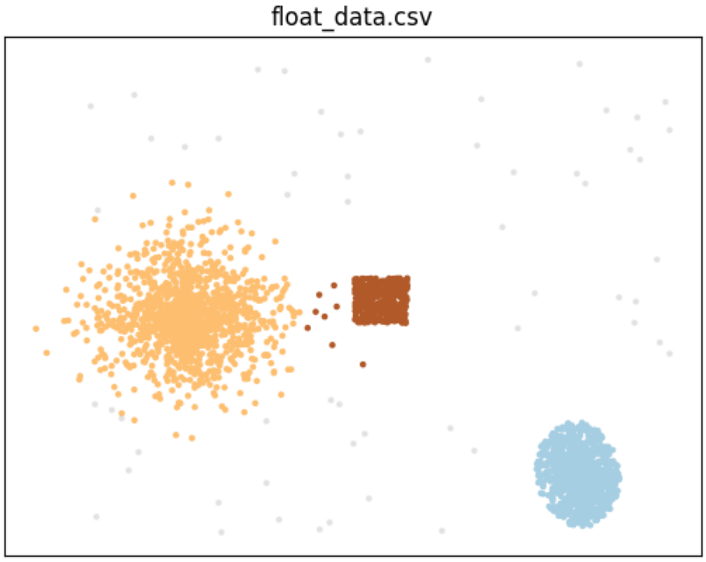

Figura 7.1: En esta imagen, se pueden observar 3 clústeres. Los puntos del mismo clúster están marcados del mismo color. Observamos que hay puntos que no pertenecen a ningún clúster (puntos de color gris) y están situados en un área distante a los tres clústeres. Estos son los puntos de anomalía que trata de identificar SDO. Generado con mdcgenpy [2].

outliers para flujos de datos. Además, clasifica algunos algoritmos existentes en algoritmos basados en distancia, en densidad y en clúster.

7.1. Sparse Data Observers

Posteriormente, tras un estudio detallado de la literatura, encontramos un estudio que se asemeja bastante a nuestro objetivo. El algoritmo propuesto se llama SDO (Sparse Data Observers) [9], que se centra en anomalías contextuales puntuales, se puede ver un ejemplo en la figura 7.1. La idea básica de este algoritmo es seleccionar representantes entre los datos, y para cada nuevo dato, se calcula la distancia euclidiana a estos representantes, que se utilizará para determinar si un punto es un *outlier* o no.

7.1.1. Descripción general

SDO consiste en dos fases: entrenamiento y aplicación. El esquema proporcionado por el artículo original del algoritmo nos da una visión muy clara de las dos fases (ver figura 7.2).

Fase de entrenamiento

Esta fase se define por tres parámetros de entrada: k, x, y q, cuyas funciones se aclararán a medida que se utilicen. Por el momento, es importante recordar que k representa el número de representantes (observadores) que el algoritmo seleccionará.

1. **Inicialización de observadores**: Se seleccionan aleatoriamente k instancias de los datos de entrenamiento como observadores iniciales.

2. **Observación de instancias**: Se calcula una matriz de distancia euclidiana (D) para cada instancia con respecto a todos los observadores. Adicionalmente, se registra a los x observadores más cercanos para cada instancia en una lista (I). Aquí se destaca el papel de x: indica el número de observadores más próximos que se conservarán para cada instancia.

3. **Eliminación de observadores ajenos**: En el paso anterior, hemos registrado los x observadores más cercanos a cada instancia. Ahora se contabiliza cuántas veces un observador aparece como *más cercano* en el vector I. Si la frecuencia de aparición de un observador es menor que q, se considera ajeno y se elimina de la lista de observadores. El valor de q actúa como un umbral que decide si un observador está suficientemente relacionado con las instancias para ser conservado. Se actualizará la lista I después de cada eliminación de observadores.

Fase de aplicación

Una vez completada la fase de entrenamiento, el modelo está preparado para su aplicación:

1. **Observación de una nueva instancia**: Se calculan las distancias entre el nuevo objeto y los observadores, almacenando los resultados en una matriz de distancias. Estas distancias se utilizarán para identificar los x observadores más próximos al nuevo objeto.

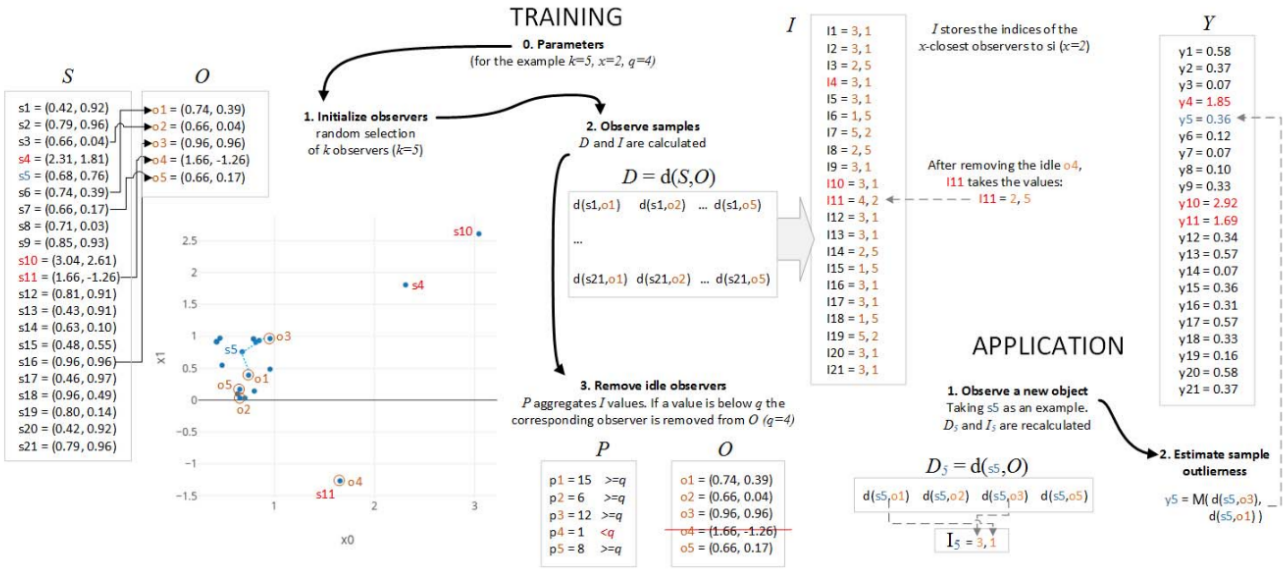

Figura 7.2: Esquema de SDO. Fuente: [9].

2. **Cálculo de la puntuación de anomalía (*outlierness*)**: Esta puntuación proporciona una medida cuantitativa del grado de anomalía de la instancia. Una puntuación más alta indica que el objeto es más anómalo. En el artículo [9], se emplea la mediana de las distancias a los x observadores más cercanos del objeto para calcular esta puntuación.

7.1.2. Análisis

Este algoritmo presenta puntos bastante destacables:

- El algoritmo opera sin supervisión, es decir, no requiere información adicional sobre si una instancia es anómala o no, incluso durante la fase de entrenamiento. Esto lo hace práctico para situaciones donde las etiquetas son escasas o inexistentes.

- Realiza una reducción significativa de los datos a un pequeño conjunto de representantes (observadores), lo que puede interpretarse como una forma de reducción de dimensionalidad. Tras esta reducción, solo es necesario almacenar los atributos de los observadores, mientras que los atributos de otros objetos

se condensan en la tabla de distancias, logrando así una eficiencia espacial notable.

- Puede detectar anomalías de forma inmediata. Para cada nueva instancia de entrada, solo necesita calcular las distancias a los observadores y realizar una operación de media o mediana, lo cual es eficiente en términos de tiempo.

A pesar de estas excelentes características, es importante ser consciente de sus limitaciones y de que no es apropiado para ciertos contextos:

- El algoritmo no diferencia puntos de diferentes clústeres; se basa en la suposición de que hay pocos puntos anómalos y muchos puntos *normales*, seleccionando así puntos que representan a estos últimos. Esto significa que la precisión del algoritmo depende de que las estimaciones estadísticas de los datos sean representativas y puedan clasificarse en representantes.

- Tiene una capacidad limitada para adaptarse a los cambios, lo cual es una consideración importante en el entorno de *data streams* donde los patrones pueden cambiar con frecuencia. La solución propuesta por el algoritmo es realizar periódicamente la fase de entrenamiento, lo cual puede no ser suficientemente dinámico para ciertos escenarios de *data streams*.

Posteriormente, se encontró una extensión de SDO [9] denominada SDOclust [10], que introduce capacidades adicionales para diferenciar puntos entre distintos clústeres. Esta variante avanzada no solo identifica anomalías contextuales puntuales, sino que también realiza *clustering* de los datos no anómalos, transformando el algoritmo de un simple detector de anomalías a un clasificador más complejo y versátil.

Además, SDOclust [10] incorpora una fase de actualización que evita la necesidad de reiniciar el proceso desde el principio durante cada ciclo de entrenamiento, permitiendo una adaptación más fluida y continua a los datos entrantes. Esta mejora representa un avance significativo en términos de eficiencia y adaptabilidad, especialmente útil en entornos de *data streams* donde las características de los datos pueden evolucionar con el tiempo.

Sin embargo, es importante destacar que, a pesar de estas mejoras, SDOclust [10] todavía opera bajo la suposición de que las estimaciones estadísticas de los datos son representativas del conjunto general. Esto implica que su eficacia puede verse limitada en escenarios donde esta suposición no se cumpla o donde la naturaleza de los datos sea altamente dinámica y heterogénea.

7.2. Streaming with Emerging New Classes

El siguiente candidato identificado se distingue por ser un planteamiento conceptual más que un algoritmo específico: se trata del problema SENC (*Streaming with Emerging New Classes*). Este enfoque se centra en la detección de nuevas clases emergentes en *data streams*. A diferencia de la detección de anomalías tradicional, SENC se enfoca en identificar y clasificar novedades, es decir, instancias de nuevas clases que emergen en el flujo de datos. Este problema plantea desafíos clave que se alinean estrechamente con los requisitos previamente identificados [11]:

- **Indeterminación de clases**: Fuera de la fase de entrenamiento, generalmente desconocemos a qué clase pertenecen los datos nuevos.
- **Inmediatez en la predicción**: Es necesario realizar predicciones en el momento en que los datos son recibidos, lo que exige una capacidad de respuesta en tiempo real.
- **Uso eficiente de memoria**: Dada la naturaleza continua y potencialmente ilimitada de *data streams*, es crucial gestionar la memoria de forma efectiva, almacenando solo los datos más relevantes.
- **Actualización rápida del modelo**: Los modelos deben ser capaces de actualizarse rápidamente para adaptarse a nuevos patrones y clases emergentes.

Estos desafíos no solo cumplen con los requisitos esenciales que hemos establecido, sino que también presentan una estrecha relación con el campo de la detección de anomalías.

Inicialmente, nos encontramos con un enfoque que extiende la aplicación de General Adversarial Network (GAN) al problema de SENC [12]. Sin embargo, optamos por no adoptarlo por varias razones. En primer lugar, el algoritmo es bastante complejo y requeriría una inmersión profunda en GAN y el aprendizaje de herramientas avanzadas como PyTorch. Además, no se disponía del código fuente del algoritmo, lo que implicaba la necesidad de una implementación completa desde cero, una tarea que excedería el alcance y el tiempo planificado para este proyecto. Afortunadamente, entre las referencias del artículo descubrimos otro algoritmo más accesible y con código fuente disponible: SENCForest [11].

7.3. La elección

Tras el análisis en las secciones anteriores, hemos identificado dos candidatos: SDO y el problema de SENC, representado por el algoritmo SENCForest. Ambos cumplen con los requisitos que hemos establecido, pero difieren en su enfoque y campo de aplicación. Tras discutirlo con el director del proyecto, decidimos continuar con SENCForest. La razón decisiva es que SENCForest nos ofrece una puerta de entrada a un campo de estudio más amplio y prometedor que es el problema de SENC.

Es importante subrayar que, aunque no profundizaremos en el algoritmo SDO, este ha sido de gran utilidad para nuestra introducción al mundo de la detección de anomalías. Además, en los experimentos que realizaremos con SENCForest, emplearemos una versión modificada del generador de datos sintéticos utilizado por los autores de SDO y SDOclust.

En el siguiente capítulo, describiremos en detalle el funcionamiento y las características del algoritmo SENCForest.

8

El algoritmo SENCForest

Este capítulo se basa en el artículo *Classification under Streaming Emerging New Classes: A Solution using Completely Random Trees*[11].

8.1. Visión general

Como se mencionó anteriormente, el enfoque principal de SENCForest, dentro del problema SENC (Streaming Emerging New Classes), es la identificación de novedades. En el contexto de SENCForest, cada nueva instancia del *data stream* es evaluada para determinar a qué clase pertenece. Si se determina que no se asemeja a ninguna clase conocida, se clasifica como una Nueva Clase Emergente (New Emerging Class o NEC). Posteriormente, estas instancias etiquetadas como NEC se incorporan al modelo para facilitar su clasificación en futuras apariciones.

SENCForest se compone de numerosos árboles de decisión. Durante la predicción de una instancia, esta es evaluada por cada árbol, y cada uno proporciona una predicción independiente. SENCForest recopila estas predicciones individuales y, basándose en la mayoría de votos, determina la clasificación final. Al igual que SDO, SENCForest requiere una fase de entrenamiento previa a la fase de predicción.

El tipo de datos que procesa SENCForest son similares a los manejados por SDO, es decir, datos que se pueden agrupar en clusters (clases). Sin embargo, a

diferencia de SDO, SENCForest no se centra en detectar anomalías externas a todas las clases, sino en identificar y clasificar nuevas clases a medida que emergen en el *data stream*. Esta capacidad implica una restricción significativa sobre el orden de aparición de las diferentes clases en el stream, lo cual se detallará en las siguientes secciones. Este aspecto será crucial al diseñar un generador de datos adecuado para SENCForest en el próximo capítulo. Por ahora, en la siguiente sección, procederemos a explicar la mecánica y el enfoque del algoritmo.

8.2. La idea del algoritmo

SENCForest es un algoritmo que se fundamenta en la premisa de que los outliers de cada clase establecen el límite máximo de lo que esa clase puede abarcar, según una distribución específica. La hipótesis es que si conocemos estos límites y una nueva instancia cae fuera de ellos, podemos inferir que esta instancia no pertenece a ninguna clase conocida y, por lo tanto, constituye una nueva clase emergente o novedad.

Para llevar a cabo esta identificación, SENCForest categoriza los datos en tres regiones distintas: normal, anómala y novedad. La región anómala, en particular, actúa como una frontera entre lo normal y lo novedoso, y está compuesta por los outliers de la clase en cuestión. Recordemos que definimos outliers como aquellos valores que, si bien son generados por la misma distribución que los valores normales, presentan una baja probabilidad de ocurrencia. Las instancias que se sitúen fuera tanto de la región normal como de la anómala serán consideradas novedades y, por ende, asignadas a la región de novedad. En la figura del artículo original, se puede observar una representación visual de estas tres regiones 8.1.

El núcleo de SENCForest son los árboles de decisión que se encargan de determinar estas regiones. Además de detectar novedades, estos árboles están diseñados para predecir a qué clase pertenecen las instancias.

Antes de detallar las operaciones específicas de SENCForest, es crucial entender cómo maneja y distingue las diferentes regiones de datos. A continuación, se responden algunos puntos claves para obtener una visión general más clara del funcionamiento del algoritmo:

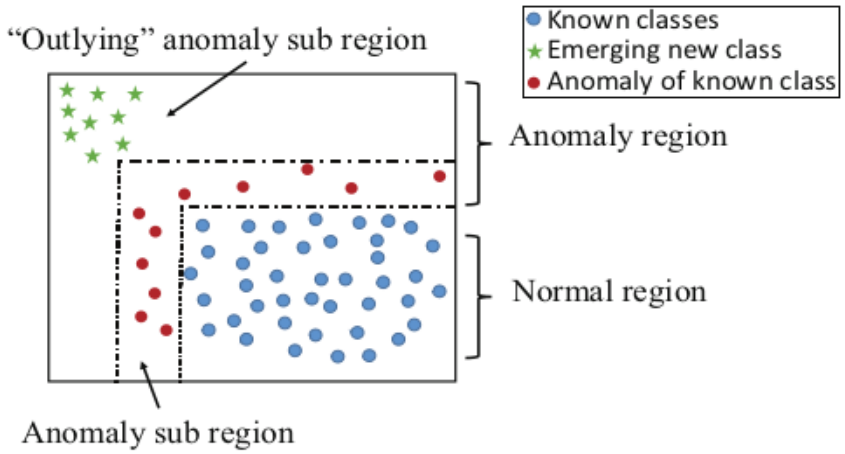

Figura 8.1: Las tres regiones de SENCForest. Fuente: [11].

- **Separación de la región normal de la anómala**

  El proceso inicia cuando una instancia de *data stream* es procesada por un árbol de SENCForest (SENCTree). La instancia recorre un camino desde la raíz hasta una hoja del árbol. Durante este trayecto, se calcula un umbral para la longitud del camino. Este umbral es crucial pues ayuda a diferenciar si una instancia cae dentro de la región normal o anómala. Los caminos más cortos suelen indicar normalidad, mientras que los más largos sugieren anomalía. Para una comprensión más visual de este proceso, se puede referir a la figura 8.1.

- **Diferenciación entre región anómala y región de novedades**

  Los SENCTrees almacenan información específica que se emplea tanto para la clasificación como para la actualización del modelo. Entre esta información, el centro y el radio de una hipersfera son fundamentales. Al calcular la distancia entre la instancia y el centro, y compararla con el radio, el algoritmo decide si la instancia se sitúa dentro de la región anómala (menor o igual que el radio) o en la región de novedades (mayor que el radio).

8.3. Entrenar un detector de clases emergentes

8.3.1. Construcción de SENCForest

La construcción de SENCForest está detallada en los Algoritmos 1 8.2 y 2 8.3, presentados en el documento en forma de pseudocódigo.

El Algoritmo 1 (Figura 8.2) se puede interpretar de la siguiente manera:

1. **Parámetros de entrada**:
   - **D**: Datos de entrada para el entrenamiento.
   - **z**: Número de SENCTree que se construirá en el conjunto de SENCForest.
   - ψ: Tamaño del subconjunto que se usará para crear cada SENCTree. En la práctica, y basándose en el código fuente, representa el número de instancias que se tomarán para cada clase existente en D. De este modo, para entrenar un SENCTree, se utilizarán en total ψ * (*número de clases en D*) instancias.
2. **Inicializar SENCForest**: Comenzar con un conjunto de SENCForest vacío.
3. **Construir SENCTrees**: Para cada árbol en el conjunto (un total de z árboles):
   - Tomar una muestra aleatoria de ψ instancias para cada clase existente en los datos de entrenamiento D. Este subconjunto Xi se utilizará para construir un SENCTree individual.
   - Añadir el SENCTree construido al conjunto de SENCForest.
4. **Resultado final**: Después de repetir este proceso z veces, tendrás un SENCForest completo, que consiste en z SENCTrees construidos a partir de diferentes muestras aleatorias de los datos de entrenamiento.

El Algoritmo 2 (Figura 8.3) describe el proceso de construcción de un SENCTree individual, que es un componente del conjunto SENCForest. Este algoritmo detalla

Algorithm 1 Build *SENCForest*

Input: D - input data, z - number of trees, ψ - subsample size.

Output: *SENCForest*

1: **initialize:** $SENCForest \leftarrow \{\}$
2: **for** $i = 1, \ldots, z$ **do**
3: $\quad X_i \leftarrow sample(D, \psi)$
4: $\quad SENCForest \leftarrow SENCForest \cup SENCTree(X_i)$
5: **end for**

Figura 8.2: Construcción de SENCForest. Fuente: [11].

cómo construir un árbol a partir de un subconjunto de datos dado, enfocándose en cómo el árbol divide los datos en cada nodo y cómo decide crear nodos hoja. Desglosemos el algoritmo paso a paso:

1. **Parámetros de entrada**:
   - **X**: Datos de entrada para el entrenamiento. En este contexto es un subconjunto de D.
   - **MinSize**: el tamaño mínimo de un nodo por debajo del cual se convierte en nodo hoja.
2. **Verificación del tamaño del nodo (creación de hojas)**: Si el tamaño del subconjunto de datos actual X es menor que MinSize, se crea un nodo hoja. Este nodo hoja almacena la siguiente información agregada sobre los puntos de datos en X que son vitales para los siguientes pasos del algoritmo:
   - El **número de instancias**(|X|)
   - Las **frecuencias de clase** (F[·]): Se obtienen contando las etiquetas asociadas a cada instancia. Esta es la única ocasión en que se utilizan las etiquetas asociadas a cada instancia hasta la fase de evaluación.
   - El **centro** (c) de la hoja: Se calcula utilizando la fórmula $c = \frac{1}{n} \sum_{x \in X} x$. Esto implica sumar todas las instancias de X y dividir por el número total de instancias.

- **Radio** (r) de la hoja: Es la distancia entre el centro y la instancia más alejada del centro. Representa el alcance máximo que las instancias tienen desde el centro y ayuda a determinar la región que abarca la hoja.

El centro y el radio se usara para distinguir la región anómala de la región de novedades.

3. **Crecimiento del árbol (División de Nodos)**: Si `X` es mayor que `MinSize`, el algoritmo procede a dividir los datos aún más:

   - Selecciona un atributo `q` al azar de la lista de atributos en `X`.
   - Elige un punto de división `p` al azar entre los valores máximos y mínimos del atributo `q` en `X`.
   - Divide `X` en dos subconjuntos: `XL` (instancias donde el atributo `q` es menor o igual a `p`) y `XR` (instancias donde `q` es mayor que `p`).

4. **Construcción recursiva del árbol**: Para cada uno de los dos subconjuntos (`XL` y `XR`), el algoritmo se llama a sí mismo recursivamente para crear nodos de rama adicionales o nodos hoja. Este proceso recursivo continúa hasta que el tamaño de los datos en un nodo es menor que `MinSize`, en ese punto se crea un nodo hoja. Los nodos que no son hojas almacenan el atributo de división (q) y el punto de división (p), que determinan la ruta que seguirá una instancia al pasar por este nodo.

5. **Salida**: La salida del algoritmo es un árbol (SENCTree) donde cada nodo interno representa una división de decisión basada en un atributo y su valor de división, y cada nodo hoja representa un subconjunto de los datos caracterizados por las divisiones realizadas en el camino hacia esa hoja.

Si has tenido experiencia en el campo de detección de anomalías o outliers, reconocerás que el Algoritmo 1 (Figura 8.2) y 2 (Figura 8.3) de SENCForest tienen similitudes con un algoritmo muy conocido en este campo: Isolation Forest (iForest). La principal distinción de SENCForest respecto a iForest reside en los nodos hoja

Algorithm 2 *SENCTree*

Input: X - input data, $MinSize$ - minimum internal node size

Output: *SENCTree*

if $|X| < MinSize$ **then**
  return LeafNode$\{|X|, F[\cdot], c, r\}$
else
  let Q be a list of attributes in X
  randomly select an attribute $q \in Q$
  randomly select a split point p from max and min values of attribute q in X
  $X_L \leftarrow filter(X, q \leq p)$
  $X_R \leftarrow filter(X, q > p)$
  return inNode{Left $\leftarrow$ $SENCTree(X_L)$,
    Right $\leftarrow$ $SENCTree(X_R)$,
    SplittAtt $\leftarrow$ q,
    SplittValue $\leftarrow$ p },
end if

Figura 8.3: Construcción de SENCTree. Fuente: [11].

de un SENCTree, donde se almacena información específica que es esencial para la clasificación de clases y la actualización del modelo.

Los nodos hoja en SENCTree guardan detalles adicionales como la frecuencia de clases, el centro y el radio de las instancias, lo que permite una mejor adaptación y respuesta ante nuevos datos o clases emergentes. Esta capacidad para ajustarse y reconocer nuevas clases es lo que lo diferencia de Isolation Forest, cuyos nodos hoja no almacenan tal nivel de detalle.

Dada la similitud estructural de SENCForest con Isolation Forest, y considerando que la principal modificación de SENCForest es en la información almacenada en los nodos hoja, es útil entender la idea básica detrás de Isolation Forest. Isolation Forest utiliza árboles binarios y está diseñado para detectar anomalías de manera rápida y eficiente tanto en tiempo (con complejidad lineal) como en uso de memoria. Opera bajo la premisa de que los puntos anómalos son más fáciles de separar o distinguir que los datos normales. Para aislar un punto de datos, el algoritmo genera particiones de forma recursiva en la muestra seleccionando aleatoriamente un atributo y, luego,

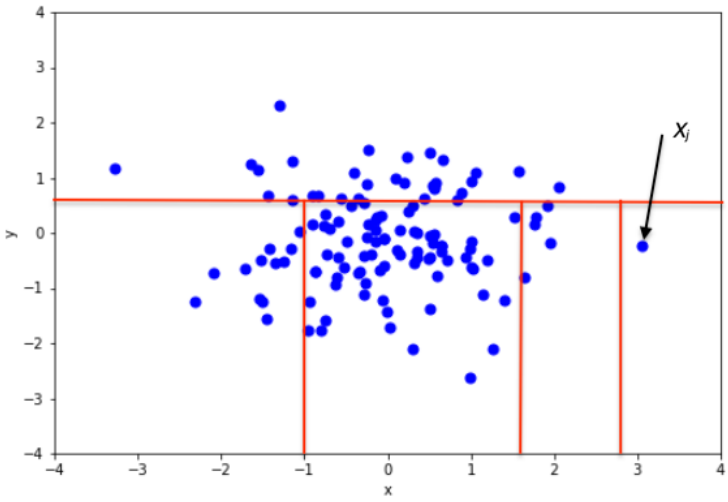

Figura 8.4: Un ejemplo de aislamiento de un punto anómalo en una distribución gaussiana 2D. Fuente: [13].

un valor de división entre los valores mínimos y máximos permitidos para ese atributo. Este proceso se repite hasta que se aísla el punto, lo que generalmente ocurre rápidamente para anomalías, puede ver un ejemplo visual en la figura 8.4.

Una vez que el modelo de SENCForest esté entrenado, necesitamos determinar para cada SENCTree un umbral que diferencie la región anómala de la región normal.

8.3.2. Determinar el umbral de longitud de camino

La longitud del camino de un nodo en un SENCTree es el número de aristas que se deben recorrer para llegar desde el nodo hasta la raíz. Para diferenciar entre las regiones anómalas y las normales dentro de un SENCTree, establecemos un umbral τ que separa estas dos regiones. Cada región en el árbol tiene su propia longitud de camino, y se espera que las regiones anómalas (A) tengan longitudes de camino más cortas en comparación con las regiones normales (K).

Para determinar el umbral de longitud de camino óptimo que separa estos dos tipos de regiones, generamos una lista L que ordena todas las longitudes de camino posibles en un SENCTree en orden ascendente. Un umbral τ en esta lista divide L en dos sub-listas: L_l y L_r. Para encontrar el mejor umbral, los autores utilizan el

siguiente criterio que minimiza la diferencia en desviaciones estándar $\sigma(.)$:

$$\hat{\tau} = \arg\min_{\tau} |\sigma(L_r) - \sigma(L_l)|$$

El umbral $\hat{\tau}$ se utiliza para diferenciar las regiones anómalas A de las regiones normales K, donde las primeras tienen longitudes de camino bajas y las segundas largas. Utilizando este método, se identifican dos regiones claramente diferenciadas: una para anomalías y otra para instancias normales.

Es importante destacar que este proceso se realiza automáticamente, sin introducir parámetros adicionales y sin requerir datos de entrenamiento etiquetados. Esto proporciona una forma eficaz y objetiva de establecer un umbral crítico para la detección de anomalías y clases emergentes en flujos de datos.

8.3.3. Separar la región anómala y la región de novedades

Una vez establecido el umbral $\hat{\tau}$, podemos diferenciar entre la región anómala y la región normal dentro de los árboles de SENCForest. Sin embargo, aún necesitamos identificar y separar la región de novedades. Como se mencionó anteriormente, en los nodos hoja de los SENCTrees, tenemos información sobre el centro y el radio del nodo, que se utiliza para construir un círculo (o hiperesferas en dimensiones más altas).

Cuando llega una nueva instancia, calculamos su distancia al centro. Si la distancia de la instancia al centro es mayor que el radio del círculo, se considera que la instancia no pertenece a ninguna de las clases conocidas y, por lo tanto, se clasifica como una instancia de clase emergente. Por otro lado, si la instancia cae dentro del círculo, se considera parte de la región anómala pero no como una novedad.

8.3.4. Del detector al clasificador

Para incorporar la función de clasificación al detector de clases emergentes, simplemente tenemos que guardar la distribución de clases de las instancias en un nodo hoja, utilizando F[j], donde j es la etiqueta que identifica una clase. En cada nodo hoja, F[j] representa la frecuencia (conteo) de instancias de cada clase que terminaron en ese nodo durante el entrenamiento. Aquí, j indexa las

clases, por lo que F[1] sería la frecuencia de instancias de la clase 1, F[2]para la clase 2, y así sucesivamente.

En la seguiente sección detallaremos cómo funciona el clasificador producido.

8.4. Aplicación en *data stream*

La implementación de SENCForest en un *data stream* se refiere a cómo se utiliza el modelo de SENCForest entrenado para clasificar instancias en datos de flujo continuo, particularmente en entornos donde pueden surgir nuevas clases. Aquí hay una explicación paso a paso del proceso de implementación del Algoritmo 3 8.5 del artículo original:

1. **Inicialización:**
   - El modelo de SENCForest entrenado, que consta de múltiples SENCTrees, está listo para su uso.
   - Se inicializa un búfer B para almacenar instancias que potencialmente provienen de una nueva clase. Este búfer tiene un tamaño predefinido s.
2. **Procesamiento de instancias transmitidas:**
   - A medida que llegan instancias en el flujo de datos, cada instancia se procesa individualmente.
   - Para cada instancia entrante x, SENCForest predice su clase.
3. **Predicción de clase:**
   - La instancia x se pasa a través de cada SENCTree.
   - Dependiendo de su trayectoria a través de los árboles y dónde cae (en términos de nodos hoja), se realiza una predicción de clase basada en votación mayoritaria o frecuencias de clase almacenadas en las hojas ($F[j]$).
4. **Manejo de instancias de posibles nuevas clases:**

- Si se predice que una instancia pertenece a una nueva clase (lo que significa que cae en una región que no coincide con las distribuciones de clase conocidas), se agrega al búfer B.
- Si B alcanza su capacidad (tamaño 's'), se activa el proceso de actualización del modelo para integrar nuevas clases en el modelo.

5. **Actualización del modelo:**

   - Una vez que el búfer está lleno, las instancias en B se utilizan para actualizar SENCForest.
   - Esta actualización puede implicar el crecimiento de nuevas ramas en los SENCTrees, ajustando la estructura existente para acomodar la nueva información, o incluso entrenando SENCTrees adicionales si es necesario.
   - Después de la actualización, el búfer se vacía y el modelo está listo para procesar nuevas instancias con su conocimiento actualizado.

Podemos ver que SENCForest utiliza un búfer para almacenar las instancias de clases emergentes y realiza una actualización cuando el búfer está lleno. Aquí hay un detalle importante que impone una limitación significativa al algoritmo: todas las instancias en el búfer se agrupan en una única nueva clase con el propósito de actualizar el modelo. Esto significa que en cada cada actualización del modelo, el algoritmo solo es posible identificar una nueva clase.

Por esta restricción, los *data streams* utilizados en los experimentos están compuestos por segmentos o porciones del flujo de datos caracterizados por una distribución de clases relativamente estable, denominados períodos. Es un marco temporal durante el cual se asume que las instancias de datos provienen de la misma distribución o una similar. Durante un período, se supone que las instancias pertenecen a un conjunto de clases conocidas o a una sola clase emergente nueva. El modelo interactúa con estos datos, intentando clasificar las instancias correctamente y detectar cualquier clase emergente.

Algorithm 3 Deploying *SENCForest* in data stream

Input: *SENCForest*, $\mathcal{B}$ - buffer of size s

Output: y - class label for each x in a data stream

```
while not end of data stream do
  for each x do
    y ← SENCForest(x)
    if y = NewClass then
      B ← B ∪ {x}
      if |B| ≥ s then
        Update (SENCForest, B)
        B ← NULL
        m ← m + 1
      end if
    end if
    Output y ∈ {b_1, ..., b_m, NewClass}.
  end for
end while
```

Figura 8.5: Aplicación de SENCForest en *data stream*. Fuente: [11].

Esta limitación sugiere que el algoritmo puede no ser eficaz para predecir múltiples nuevas clases en un solo período. También influirá en el diseño del generador de datos sintéticos que implementaremos para los experimentos, ya que deberá tener en cuenta la estructura de periodos.

En la siguiente sección, explicaremos cómo se actualiza el modelo teniendo en cuenta esta restricción.

8.5. Actualización del modelo

Hay dos mecanismos para mantener el modelo actualizado:

8.5.1. Crecimiento de un subárbol en un SENCTree

Actualizar SENCForest con el búfer B es un proceso que actualiza cada nodo hoja en cada árbol utilizando ψ instancias seleccionadas aleatoriamente de B. Esto se describe en el Algoritmo 4 8.6, explicandolo paso a paso:

1. **Inicialización:**

   - Todas las instancias en el búfer B se asignan inicialmente a una nueva clase, b_{m+1}. Este paso asume que las instancias en B representan potencialmente una nueva clase no vista antes.

2. **Actualizando cada árbol:**

   - Para cada árbol ($i = 1$ a z) en el SENCForest:
     - Se selecciona aleatoriamente un subconjunto B' de tamaño ψ del búfer B.
     - Para cada nodo hoja j en el árbol actual:
       - Identificar instancias X' de B' que caen en el nodo hoja j.
       - Si X' no está vacío (es decir, hay instancias de la nueva clase que caen en esta hoja), proceder a actualizar el nodo hoja.

3. **Actualizando nodos hoja:**

   - Para cada nodo hoja que requiere actualización:
     - Generar pseudo instancias X que representan los datos existentes en el nodo hoja. Estas pseudo instancias se basan en las características de datos existentes del nodo hoja (como el centro c y el radio r).
     - Combinar pseudo instancias X con nuevas instancias X' del búfer.
     - Cultivar un nuevo subárbol usando este conjunto de datos combinados $X' \cup X$, reemplazando efectivamente el viejo nodo hoja con este nuevo subárbol.

4. **Recalcular el umbral de longitud del camino:**

   - Después de actualizar cada árbol, recalcular el umbral de longitud del camino $\hat{\tau}$ para el árbol siguiento el método definido en la sección 8.3.2. Este umbral necesita ser actualizado para reflejar los cambios realizados en el árbol.

Algorithm 4 Update *SENCForest*

Input: *SENCForest* - existing model, $\mathcal{B}$ - input data

Output: a new model of *SENCForest*

initialize: All instances in $\mathcal{B}$ are assigned a new class b_{m+1}
for $i = 1, ..., z$ **do**
  $\mathcal{B}' \leftarrow sample(\mathcal{B}, \psi)$
  Tree $\leftarrow$ *SENCForest*.Tree$[i]$
  for $j = 1, ...,$Tree.LeafNodeNumber **do**
    $X' \leftarrow$ instances of $\mathcal{B}'$ which fall into Tree.LeafNode$_j$
    if $|X'| > 0$ **then**
      $X \leftarrow$ Pseudo instances from Tree.LeafNode$_j$
      $X' \leftarrow X' \cup X$
      Tree.LeafNode$_j \leftarrow SENCTree(X')$
    end if
  end for
  recalculate $\hat{\tau}$ for Tree
  SENCForest.Tree$[i] \leftarrow$ Tree
end for

Figura 8.6: Actualización de SENCForest. Fuente: [11].

5. **Salida final:**

   - La salida es un modelo SENCForest actualizado, donde cada árbol ha sido ajustado para incorporar las nuevas instancias de clase del búfer B.

Del Algoritmo 4 8.6, la parte clave se concentra en la actualización de los nodos hoja 3. Aquí vemos otra función de las cuatro variables ($|X|$, $F[.]$, c, r) que se guardó en los nodos hoja, explicado en la Creación de Hojas 2 del Algoritmo 2 8.3. Con estas cuatro variables podemos generar $|X|$ número de puntos de datos, siguiendo una distribución de clases $F[.]$ dentro de las circunferencias (o hiperesferas para dimensiones más altas) definidas por el centro c y el radio r. Estos puntos de datos son los que en el Algoritmo 4 8.6 se llaman *Pseudo instancias*. Estas *Pseudo instancias*, junto con las instancias nuevas, se pasan por el Algoritmo 2 8.3 (Construcción de SENCTree) generando un subárbol que substituirá al nodo hoja del árbol de partida.

8.5.2. Crecimiento de múltiples SENCForest

Recordemos que al principio fijamos el número de SENCTrees (parámetro z) que debe tener un SENCForest (Algoritmo 1 8.2). Cuando el número de clases existentes en un SENCForest alcanza cierto límite p establecido por el usuario, los SENCTrees de este último dejan de actualizarse para cualquier nueva clase emergente. En su lugar, se construirá un nuevo SENCForest para las siguientes p clases emergentes.

8.5.3. Predicción usando múltiples SENCForest

La predicción final al tener múltiples SENCForest es la siguiente:

1. Cada SENCForest en el conjunto ofrece su predicción para una instancia dada. Esta predicción puede ser una etiqueta de clase conocida o una indicación de que la instancia pertenece a una nueva clase.

2. Se combinan estas predicciones para decidir el resultado final:

   - Si todos los *SENCForests* predicen que la instancia pertenece a una nueva clase (es decir, todos proporcionan como salida "NuevaClase"), entonces la predicción final es "NuevaClase". Este enfoque de consenso ayuda a reducir los falsos positivos en la detección de novedades, ya que solo se considera nueva si todos los bosques están de acuerdo, asumiendo que han sido entrenados en diferentes subconjuntos de clases.

   - Si algún SENCForest predice que la instancia pertenece a una clase conocida, entonces estas predicciones se consideran para la salida final. La clase específica conocida con la mayor probabilidad a través de los bosques es elegida como la predicción final. La probabilidad para una clase específica se calcula como sigue:

$$p_i = \frac{\text{Número de SENCTrees que predicen } y_i}{\text{Número total de SENCTrees}},$$

     donde y_i es la predicción del SENCForest i.

8.5.4. Mecanismo de liberación de memoria

Uno de los objetivos del problema SENC es la eficiencia espacial. Para lograrlo, SENCForest requiere un mecanismo que libere memoria de modelos que ya no son eficientes, ya sea por la evolución del flujo de datos o por cualquier otra razón. El algoritmo plantea un mecanismo para retirar SENCForest a medida que avanza el flujo de datos. Un SENCForest se retira bajo los siguientes escenarios:

1. Cuando un SENCForest no se utiliza para predecir clases conocidas durante un cierto período de tiempo, se elimina para cualquier predicción futura. En otras palabras, si un SENCForest emite "NuevaClase" durante mucho tiempo, este SENCForest será retirado.

2. En caso de que el número de SENCForests haya alcanzado el límite preestablecido ρ y ningún SENCForest pueda ser retirado según lo anterior, entonces se opta por retirar el SENCForest menos utilizado en el último período. Para identificar al SENCForest menos utilizado, se revisa el número de predicciones registradas para cada uno en el flujo de datos. El SENCForest que haya realizado el menor número de predicciones para clases conocidas es considerado el menos utilizado y, por lo tanto, es el elegido para ser retirado.

8.6. Reflexiones finales

Hasta aquí hemos concluido la explicación teórica del SENCForest. En mi opinión, aunque posea restricciones significativas que limitan su aplicación en datos reales, la concepción del algoritmo es notablemente innovadora y me ha sorprendido con varios de sus detalles técnicos. Es evidente que SENCForest aporta una perspectiva única y soluciones creativas a la detección de clases emergentes en *data stream*, lo que lo convierte en un punto de referencia interesante para futuras investigaciones en este campo. Además, sus mecanismos de actualización y retiro de modelos añaden una capa de adaptabilidad y eficiencia que merecen ser exploradas y potencialmente mejoradas para ampliar su aplicabilidad.

En el siguiente capítulo, avanzaremos hacia la fase de desarrollo, empezando por la implementación de un generador de datos aplicable a SENCForest, el cual nos proporcionará los datos necesarios para los experimentos.

9

DAGADENC: Un generador de datos sintéticos para algoritmos de detección de clases emergentes en data streams

En este capítulo describimos en detalle nuestro generador de datos sintéticos diseñado para el estudio de algoritmos de detección de clases emergentes en data streams. Este generador es bautizado como DAGADENC (DAtaset Generator for Algorithms for Detection of Emerging New Classes).

Esta herramienta es uno de los aportes originales de nuestro trabajo. Está específicamente desarrollado para producir datos en el formato de entrada de SENCForest, pero puede utilizarse para experimentar con cualquier algoritmo de detección de anomalías, y más concretamente aquellos enfocados a la detección de clases emergentes, eventualmente con alguna modificación menor que adapte los datos generados al formato de entrada del algoritmo que deseamos estudiar.

9.1. Requisitos del generador

Como se mencionó en el capítulo anterior, los datos para la experimentación de SENCForest están compuestos por una secuencia de periodos. En un periodo, se asume que las instancias pertenecen a un conjunto de clases conocidas o a una sola clase emergente nueva. En la práctica, cada periodo se supone una ejecución

del algoritmo SENCForest. Con el modelo resultante de la ejecución, se puede iniciar la ejecución del siguiente periodo.

El formato del *data stream* puede ejemplificarse con una demostración utilizada por los autores. Supongamos que tenemos n periodos representados como $t_0, t_1, \ldots, t_{n-1}$. t_0 es el primer periodo que se pasa como entrada al algoritmo. Como es el primer periodo, se asigna a la fase de entrenamiento para construir el modelo de SENCForest. En los experimentos de los autores, para el periodo t_0 se utilizan muestras con instancias de dos clases para entrenar el modelo. Una vez finalizada la fase de entrenamiento, pasamos al periodo t_1; a partir de aquí ya se puede aplicar el modelo entrenado. Los periodos en la fase de aplicación contienen instancias de tres clases diferentes: dos clases ya conocidas de periodos anteriores y una nueva clase emergente que el algoritmo deberá identificar. Durante cada aplicación, el modelo se irá actualizando y adaptando al flujo de datos sin necesidad de pasar por otra fase de entrenamiento. Además, es importante que los periodos tengan un número de instancias similar para no afectar la evaluación del resultado, con la excepción del periodo t_0, que se usa para entrenar el modelo, no para la aplicación.

9.2. El generador escogido

En la sección 7.3, mencionamos que aunque no seleccionamos SDO [9] como nuestro algoritmo de estudio principal, descubrimos que los autores de SDO utilizaron un generador de clústeres para sus experimentos que se ajusta bastante a lo que buscamos. Con algunas modificaciones, podemos adaptarlo al tipo de datos necesario para los experimentos de SENCForest.

La base teórica del generador está presentada en el artículo [14]. El generador presentado en el artículo, llamado MDCStream, es para conjuntos de datos numéricos dependientes del tiempo, con el fin de probar algoritmos de clasificación, *clustering* y detección de anomalías en datos de flujo; es decir, además de generar puntos de datos, añade una dimensión temporal. Sin embargo, para este proyecto, lo que nos interesa es el generador de clústeres que utiliza: MDCGen. Inicialmente, el código de MDCGen está escrito en Matlab, y dado que el desarrollador no tiene mucha

experiencia en el entorno de Matlab, hemos utilizado una versión en Python hecha por uno de los autores del artículo, conocida como MDCGenpy [2]. A continuación, presentaremos el funcionamiento de MDCGenpy el cual es la base para el desarrollo de nuestro generador DAGADENC.

9.3. MDCGenpy

Resumiremos brevemente los pasos originales de *MDCGenpy*:

1. **Configuración de parámetros**

   - *Validación de parámetros de entrada:* Revisión y validación de la consistencia de los parámetros introducidos, incluyendo número de instancias, número de dimensiones, número de clústeres, distribuciones de probabilidades usadas para generar los clústeres, entre otros. Para una lista completa de parámetros, se puede consultar la documentación del generador [15].
   - *Inicialización de parámetros de configuración:* Tras validar, se inicializan todas las variables globales y se almacenan en el objeto `clus_cfg`.

2. **Generación de masa de clústeres (generate_mass)**

   - **Objetivo:** Determinar el número de muestras a generar para cada clúster, utilizando la información de `clus_cfg` para asegurar que el número total de muestras alcance el número especificado.
   - **Entrada:** `clus_cfg`.
   - **Retorno:** Un array `mass` con el número de muestras para cada clúster.
   - **Proceso:**
     - Si en `clus_cfg` no hay una configuración definida para el número de instancias por clúster, se genera un `mass` array aleatorio usando una distribución uniforme entre 0 y 1, se normaliza para que la suma total sea igual al número especificado de muestras y se ajusta para cumplir con el mínimo de muestras por clúster.

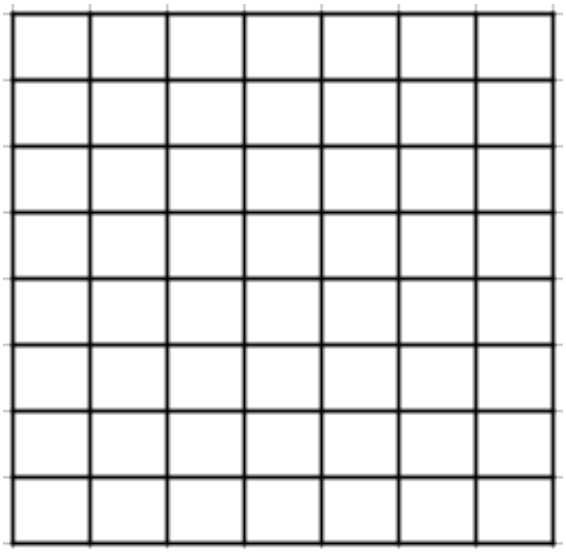

Figura 9.1: Ejemplo visual de una cuadrícula de dos dimensiones. Fuente: Elaboración propia.

3. **Generar posiciones para el centroide de los clústeres: locate_centroids(clus_cfg)**

   Esta función es una de las más difíciles de entender del programa. Para comprender su funcionamiento, es necesario referirse a la explicación del artículo.

   Antes de crear cualquier punto de datos, el generador necesita dividir el espacio en una especie de cuadrículas. Esto es para determinar dónde ubicar los clústeres, ya que la distancia entre ellos debe ser configurable para los experimentos. Además, recuerde que este generador se usa para SDO, por lo que es importante mantener a los *outliers* distantes de todos los clústeres. Para determinar esta cuadrícula, para cada dimensión calcula la densidad de la cuadrícula a_i:

$$a_i = 2 + C_i \left(\frac{1 + \ln(n_{\text{Clusters}})}{n_{\text{Clusters}}} + \frac{\ln(n_{\text{Outliers}})}{n_{\text{Outliers}}} \right)$$

   Donde a_i es el número de hiperplanos equidistantes que dividen en una dimensión. Por ejemplo, en dos dimensiones, si a_i es [8, 8], el espacio se dividiría en una cuadrícula de 8x8 9.1.

Una vez determinada la cuadrícula, se procede al posicionamiento de los centros de los clústeres. Como cada celda de la cuadrícula es equidistante, los centros se asignan de manera aleatoria a las intersecciones de la cuadrícula. Más tarde se introduce un ruido en cada centro para evitar que estén perfectamente equidistantes entre sí.

4. **Generar clústeres**

   Esta sección aborda la generación de clústeres y define los preparativos para la creación de lotes de datos, culminando con la llamada a `compute_batch`.

   - *Generación de matriz de transformación*: Proporciona la posibilidad de establecer correlaciones entre los atributos de los datos y aplicar rotaciones a los puntos. Esto permite generar clústeres con características específicas y variabilidad en su orientación y distribución.
   - *Generación de datos*: Hasta aquí ya tenemos los preparativos listos para la generación de los puntos de datos, procedemos a la llamada `compute_batch` que se ocupa de la generación.

5. **Generación de datos con compute_batch**

   Los procesos dentro de esta función son los siguientes:

   a) Basándose en el número de muestras para cada clúster, calcula la probabilidad $p[i]$ para cada uno, donde $p[i] = \frac{\text{número de muestras para el clúster } i}{\text{número total de muestras}}$. Luego genera una lista aleatoria del tamaño del número total de muestras. Los elementos de esta lista son las etiquetas de los clústeres, distribuidos según las probabilidades p. Esta lista determinará el orden de generación de los datos. El objetivo de cambio es que esta lista esté ordenada siguiendo la estructura de periodo que necesitamos para nuestros experimentos.

   b) Genera los datos en el orden especificado anteriormente. Los datos de cada clúster se generan siguiendo la distribución de probabilidad definida por los parámetros de configuración. Se aplican las matrices de correlación

y rotación, y cada punto de datos se desplaza por el centroide del clúster, moviendo efectivamente todo el clúster a su posición correcta en el espacio de características.

c) Generación de *outliers*. Este paso no se usará para nuestro generador modificado, ya que en el contexto de detección de clases emergentes, un *outlier* de este tipo no se puede clasificar en ninguna clase.

Mantendremos la lógica general de generación de datos, pero ajustaremos la probabilidad y el orden de generación de los clústeres para alinearlos con los objetivos específicos de nuestros experimentos.

9.4. DAGADENC: Adaptando MDCGenpy para la detección de clases emergentes

En esta sección explicaré las modificaciones hechas al generador original para que genere datos para el problema de decocción de clases emergentes, y en concreto con en un formato apto para ser procesado con SENCForest.

En la sección anterior, observamos que el generador original produce el orden de generación de los datos siguiendo una lista que contiene etiquetas de las clases a generar, generada aleatoriamente en función del número de muestras para cada clúster. Nuestro objetivo es generar una lista similar, pero con un orden de etiquetas que se ajuste a nuestra estructura de períodos. Introduciremos dos mecanismos para generar esta secuencia de etiquetas.

En un período, los clústeres que han aparecido en períodos anteriores serán denominados clústeres conocidos, y los clústeres que no han aparecido hasta el período actual serán denominados clústeres nuevos. Considerando que para todos los períodos tenemos un número fijo de instancias $n\_samples\_per\_period$, estas instancias pertenecen a $n\_old\_cluster + n\_new\_cluster$ clústeres diferentes. Aquí, $n\_old\_cluster$ representa el número de clústeres conocidos y $n\_new\_cluster$ el número de clústeres nuevos (en nuestros experimentos será uno) que el algoritmo debe

reconocer en este período. El primer período es la excepción, ya que corresponde a la fase de entrenamiento y todas las instancias en este período serán de clústeres nuevos.

1. **Mecanismo 1: Distribución de clústeres en períodos** Como se explicó anteriormente, la función `generate_mass` retorna un array `mass` con el número de muestras para cada clúster. Con este array `mass`, podemos calcular la proporción $p[i]$ que ocupa cada clúster en el total de las muestras. Esta $p[i]$ participa directamente en determinar el orden de generación de las instancias. Sin embargo, debido a la estructura de períodos, no podemos aplicar este array directamente para determinar el número de instancias a generar. Expliquemos nuestra solución con un ejemplo. Supongamos que los clústeres A, B, y C están asignados al período actual:

   - Sumaremos el total de muestras que ocupan los clústeres asignados en el array `mass`, es decir, $sum = mass[A] + mass[B] + mass[C]$. Luego, calcularemos la proporción que ocupa cada clúster; por ejemplo, para el clúster A: $p[A] = mass[A]/sum$. Esta proporción se multiplicará por el número de muestras por período: $muestras\_A = p[A] * n\_samples\_per\_period$. De esta manera, conseguimos un número de muestras para cada clúster ajustado al período actual, manteniendo la proporción original definida en `mass`.

2. **Mecanismo 2: Incorporación de clústeres nuevos y eliminación de clústeres viejos**

   Definimos como clúster *vivo* aquel que sigue activo en el flujo de datos y puede ser asignado a nuevos períodos, por el contrario, consideramos que está *muerto* o *destruido*. Antes de procesar el primer período, se inicia un diccionario `clusters_live_probability` que almacena la probabilidad de supervivencia de los clústeres del período actual, asignando a cada clúster una probabilidad uniforme inicial: 1/número de clusters. Antes de avanzar al siguiente período, se llevarán a cabo los siguientes pasos:

a) **Selección y eliminación de un clúster *muerto*:** Se seleccionará un clúster para *morir* según sus probabilidades de supervivencia. Un clúster *muerto* no será asignado a ningún otro período posterior. Una vez elegido el clúster víctima, se eliminará su probabilidad del diccionario `clusters_live_probability`, dejando una probabilidad *libre* para ser redistribuida.

b) **Decremento de la probabilidad de supervivencia de los clústeres supervivientes:** Para simular un envejecimiento natural, se disminuirá la probabilidad de supervivencia de los clústeres supervivientes. Los clústeres más *viejos* tendrán mayor probabilidad de *morir*. Este decremento se calcula de la siguiente manera: Sea $p_i = \text{clusters\_live\_probability}[i]$ la probabilidad de supervivencia del clúster i, al finalizar el período, p_i se reducirá a $p_i = p_i \cdot (1 - \text{tasa de decremento})$. Donde la tasa de decremento $= c \cdot \text{dead\_factor}$, siendo c una constante. En nuestro caso, hemos configurado la constante c con el valor 0.2, y el valor de `dead_factor` debe estar entre 0 y 4 para un funcionamiento correcto. Estos dos valores pueden ser modificados según las necesidades del usuario.

El proceso anterior es aplicable para los períodos posteriores al período 1. En el caso del período 1, la lógica es similar, pero no hay que eliminar ningún clúster. Como para el período 2, ningún clúster *muere* y solo se incorporan n clústeres nuevos, se realiza una normalización y se libera una probabilidad: $free\_probability = (1/\text{period\_clusters}) \cdot \text{clus\_cfg.n\_new\_cluster}$.

Finalmente, la probabilidad que se libera, ya sea en el caso particular del período 1 o en otros períodos, se repartirá uniformemente entre los n clústeres nuevos del siguiente período.

9.5. Resultado

Después de modificar el generador, este producirá datos siguiendo una secuencia que respeta el orden de los períodos establecidos. Cada período está representado por

una serie de instancias determinadas, y estas series se ordenan de manera secuencial para reflejar la progresión de los períodos. Por ejemplo, el primer período contiene las primeras instancias, el segundo período las siguientes, y así sucesivamente, hasta cubrir todos los períodos definidos.

Además de esta secuenciación temporal en la generación de los clústeres, se conservan las funcionalidades originales del generador. Estas incluyen la capacidad de establecer correlaciones entre clústeres, seleccionar diversas funciones de distribución para la generación de clústeres, aplicar rotaciones, ajustar la distancia entre los clústeres, y más. Estas características se detallan en la documentación de MDCGenpy [15].

Cabe destacar que, dado que el código original del generador se desarrolló en inglés, hemos mantenido este idioma para nuestro trabajo de desarrollo. Por tanto, es posible que encuentres gráficos o tablas en inglés en la presentación de los datos generados. Lo mismo pasa como en los experimentos realizados con SENCForest.

En esta sección, experimentaremos con el generador modificado y mostraremos algunos conjuntos de datos bidimensionales, lo que nos permite representarlos gráficamente para un efecto más visual.

Para una presentación más clara de los conjuntos de datos, se mostrarán dos imágenes para cada uno. La primera imagen, marcada con “A”, exhibirá el conjunto de datos en un gráfico, ofreciendo una visión general de los clústeres. La segunda imagen, marcada con “B”, representará los datos separados por períodos, permitiendo observar la aparición y desaparición de clústeres a medida que avanzan los períodos.

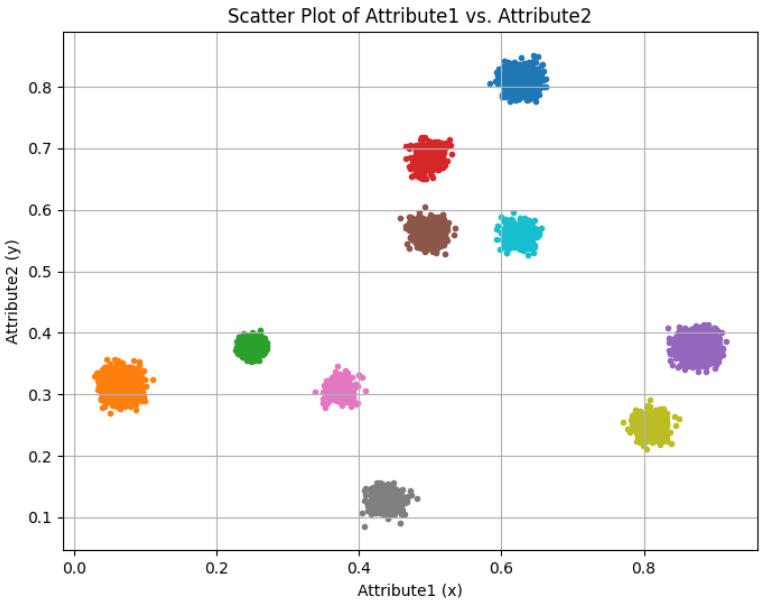

Figura 9.2: Data stream 1-A. Sin solapamiento entre clústeres. Distribución gaussiana. Fuente: generado con DAGADENC [3].

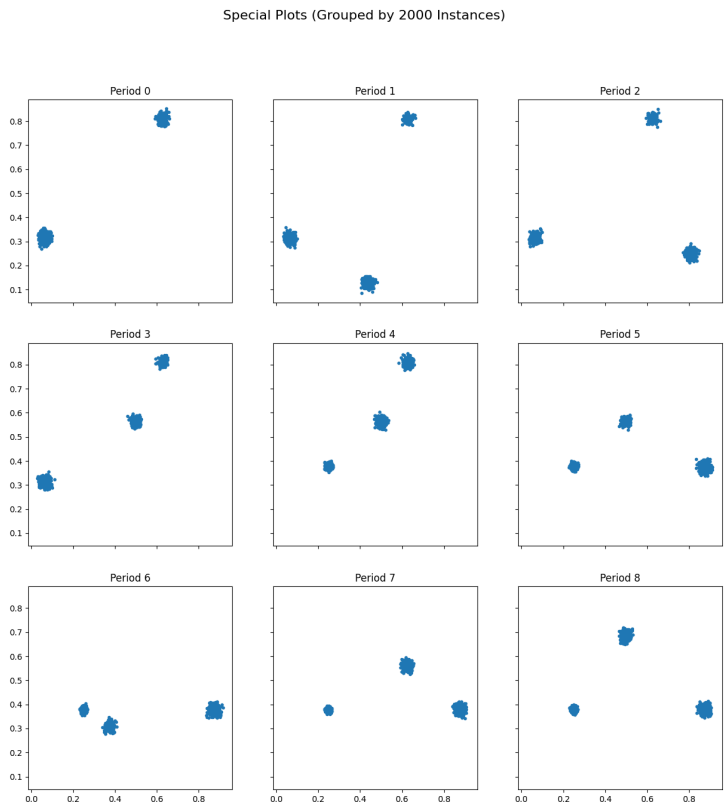

Figura 9.3: Data stream 1-B.Fuente: generado con DAGADENC [3].

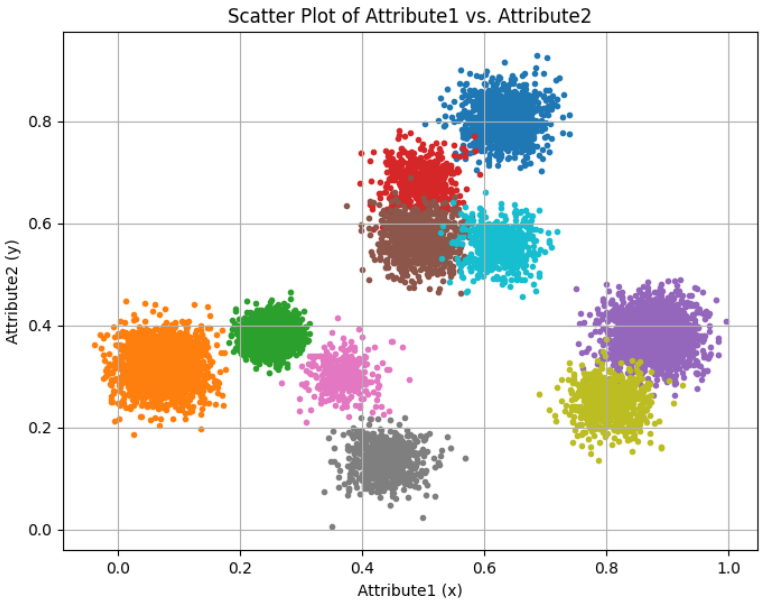

Figura 9.4: Data stream 2-A. Con solapamiento entre clústeres. Distribución gaussiana. Fuente: generado con DAGADENC [3].

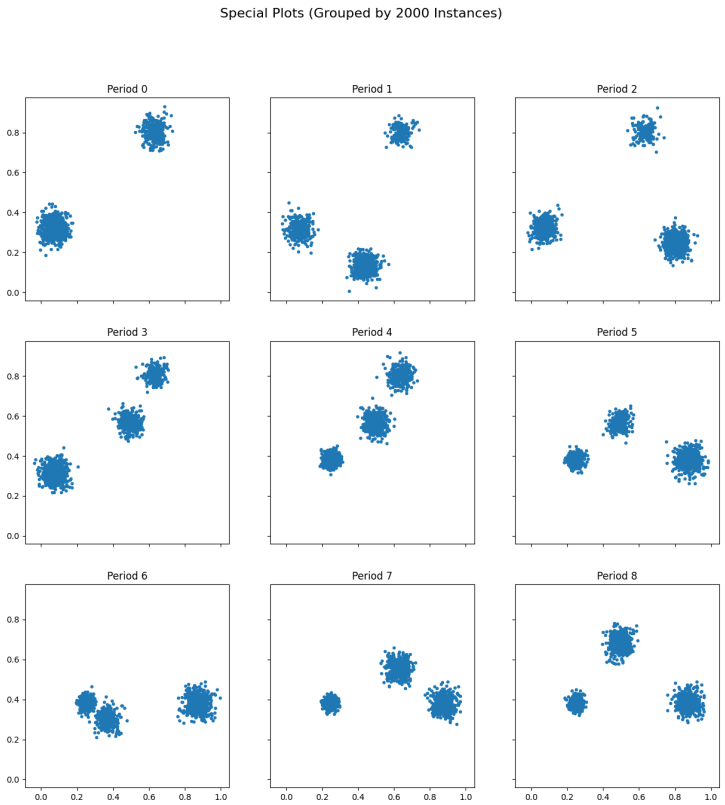

Figura 9.5: Data stream 2-B.Fuente: generado con DAGADENC [3].

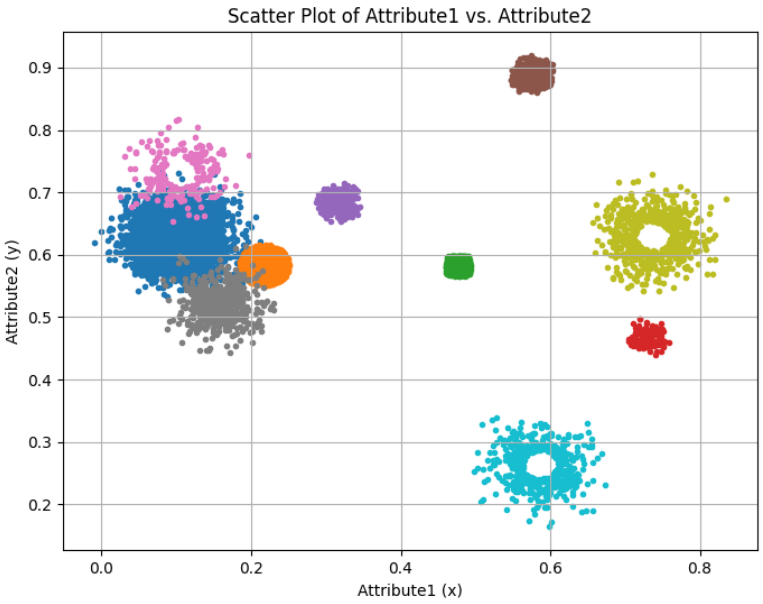

Figura 9.6: Data stream 3-A. Con solapamiento entre clústeres. Distribución gaussiana, triangular, gap y uniforme. Fuente: generado con DAGADENC [3].

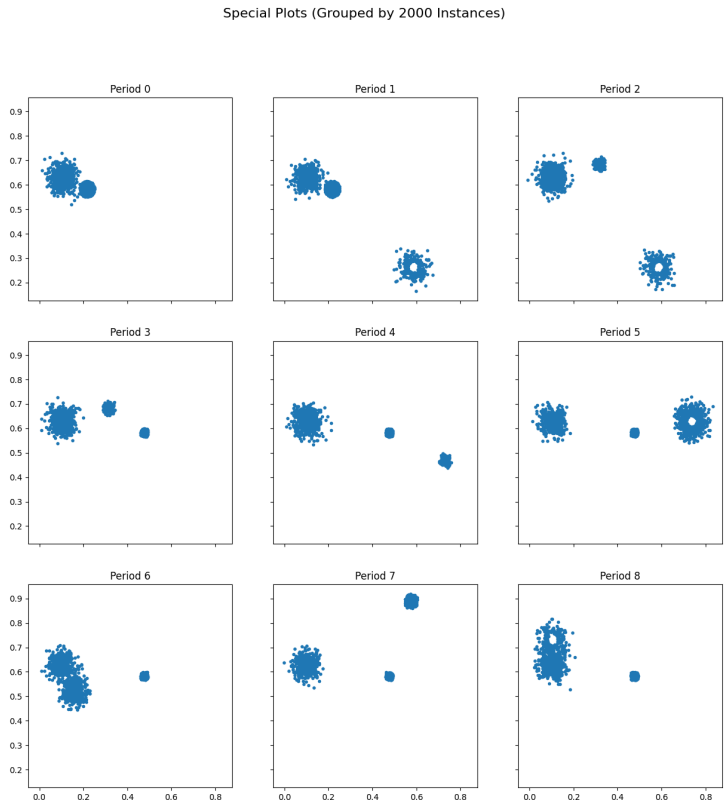

Figura 9.7: Data stream 3-B.Fuente: generado con DAGADENC [3].

10 Experimentos

En este capítulo presentaremos como hemos hecho los experimentos y los resultados de los experimentos. Seguiremos la configuración de los autores de SENCForest presentado en el artículo [11]. Asimismo, describimos las diferentes medidas de evaluación del rendimiento y calidad de algoritmos de detección de clases emergentes, y nuestra implementación del software necesario para realizar tales medidas y analizar los resultados obtenidos. El estudio experimental y el desarrollo de algunas herramientas para la evaluación de algoritmos de detección de clases emergentes es otro de los aportes originales de nuestro trabajo.

10.1. Medidas de evaluación

En los experimentos de SENCForest, se utilizan dos medidas principales de evaluación para valorar el rendimiento del modelo: EN Accuracy y F-measure.

10.1.1. EN Accuracy (Emerging New Accuracy)

EN Accuracy refleja la capacidad general del modelo para clasificar correctamente instancias tanto de clases conocidas como nuevas emergentes. Se calcula con la siguiente fórmula:

$$\text{EN\_Accuracy} = \frac{A_n + A_o}{N}$$

Donde:

- A_n: Número total de instancias de la nueva clase emergente identificadas correctamente.
- A_o: Número total de instancias de la clase conocida clasificadas correctamente.
- N: Número total de instancias en la ventana de evaluación.

Una EN Accuracy más alta indica un mejor rendimiento general, con el modelo reconociendo efectivamente tanto las clases existentes como adaptándose a las nuevas.

10.1.2. F-measure

F-measure (o F1 *score*) es una métrica ampliamente utilizada en tareas de clasificación, especialmente en situaciones donde los datos están desequilibrados, como ocurre con los datos generados por nuestro generador, en los que la proporción ocupada por cada clase varía en cada periodo. F-measure representa la media armónica de la precisión y el recall, ofreciendo un equilibrio entre ambas.

Antes de definir precisión y recall, tenga en cuenta lo siguiente:

- **True Positive (TP)**: número de instancias identificadas correctamente como de la clase nueva.
- **False Positive (FP)**: número de instancias etiquetadas incorrectamente como de la clase nueva cuando no lo eran.
- **False negative (FN)**: número de instancias que eran realmente de la clase nueva, pero no fueron identificadas como tal.

Precision (P):

La proporción de instancias positivas identificadas correctamente sobre el total de instancias identificadas como positivas. Para la detección de nuevas clases, sería

la proporción de instancias de la nueva clase identificadas correctamente sobre todas las instancias identificadas como nueva clase. Se calcula con la siguiente fórmula:

$$Precision = \frac{TP}{TP + FP}$$

Una alta Precision indica que el algoritmo devolvió sustancialmente más resultados relevantes que irrelevantes. En el contexto de SENCForest, una alta Precision significa que cuando predice una instancia como una clase nueva, es probable que sea correcto.

Recall (R):

La proporción de instancias positivas identificadas correctamente sobre el total de instancias positivas reales. En el contexto de detección de nuevas clases, es la proporción de instancias de la nueva clase identificadas correctamente sobre todas las instancias reales de la nueva clase. Se calcula de la siguiente fórmula:

$$Recall = \frac{TP}{TP + FN}$$

Un alto Recall indica que el algoritmo devolvió la mayoría de los resultados relevantes. En el contexto de SENCForest, un alto Recall significa que identifica exitosamente la mayoría de las instancias de la clase nueva.

F-mesure se calcula con la siguiente fórmula:

$$F\text{-measure} = 2 \cdot \frac{P \cdot R}{P + R}$$

Una F-measure alta indica que el algoritmo identifica efectivamente nuevas clases con un buen equilibrio de precisión y recall. Una F-measure de 1 indica precisión y recall perfectos, mientras que una puntuación más cercana a 0 indica un rendimiento deficiente.

Al igual que el generador de datos, el algoritmo SENCForest está implementado en inglés. Por lo tanto, he mantenido el idioma original en las pequeñas modificaciones que realicé al código. Entre los cambios implementados, se incluyen la adición de cálculos para la EN Accuracy y la F-measure, que estaban descritos en el

artículo del algoritmo, pero no estaban implementados en el código de SENCForest Además, se ha implementado otro script principal, Main.m, para que trabajara con los datos generados por la versión modificada del generador mdcgenpy. Todos los códigos fuente, junto con los conjuntos de datos utilizados en los experimentos, están disponibles en el repositorio de Github [3].

10.2. Objetivos del experimento

Después de revisar los experimentos realizados en el artículo [11], he identificado los siguientes comportamientos interesantes del SENCForest para probar:

1. Comparación de resultados para datos con clases aisladas (ejemplo: 9.2) y comportamiento en datos con clases superpuestas (ejemplo: 9.4).

2. Comparación en datos de diferentes dimensiones, incluyendo la comparativa de métricas EN_Accuracy y F-measure. También examinaremos cómo varía el tiempo de ejecución al incrementar la dimensión de los datos.

3. Análisis en data streams largos, con el objetivo de evaluar la eficiencia del Model Update de SENCForest a lo largo de múltiples períodos.

Podemos abordar los objetivos 1 y 2 con datos de dos períodos (uno de entrenamiento y otro de aplicación). Para el objetivo 3, necesitamos data streams con varios períodos.

Tras probar el script de SENCForest publicado por los autores [16], noté un defecto en la actualización del modelo durante el proceso de aplicación. En un funcionamiento correcto de la actualización del modelo, cuando el algoritmo detecta que el buffer de instancias de la nueva clase está lleno, debería proceder a la actualización del modelo actual. Sin embargo, observé que el algoritmo ejecuta el código de actualización, pero no guarda el resultado en el modelo actual. Esto provoca que el modelo no se actualice y que el algoritmo detecte instancias de clases que ya deberían estar actualizadas como clases emergentes. He intentado buscar la fuente del problema y solucionarlo, pero no tuve mucho éxito, posiblemente debido

a las reglas de paso por referencia o valor en Matlab, el lenguaje en el que está escrito el código fuente de SENCForest. El algoritmo almacena el Modelo en un tipo *struct*, que por defecto se pasa por valor, lo que implica que modificaciones dentro de la función no afectan al Modelo original.

Intenté modificar la función para que retornara el Modelo actualizado y así asignarlo al Modelo original, pero no hubo cambios.

Debido a esta incidencia, decidí no realizar los experimentos del punto 3. Estoy seguro de que con más tiempo y un estudio más profundo de Matlab y el código fuente de SENCForest, podría resolver el problema. Sin embargo, en esta fase de los experimentos, no dispongo del tiempo necesario para emprender este estudio. Esta incidencia fue bastante sorprendente y, como pueden ver en la sección de Riesgos 3.2, nos enfocamos en los riesgos durante la fase de exploración de algoritmos, sin dar mucha importancia a otros posibles riesgos como este.

En las siguientes secciones presentaremos los resultados obtenidos de las ejecuciones de datos. Puede encontrar los datos y el código fuente utilizado para este proyecto en Github [3]. Tenga en cuenta que, debido al carácter aleatorio del algoritmo, es posible obtener resultados diferentes incluso bajo las mismas condiciones y ajustes del entorno. Por esta razón, las métricas presentadas corresponden al promedio de cinco ejecuciones realizadas en condiciones idénticas. Se usarán los parámetros de algoritmos definidos en el artículo original [11].

10.3. Configuración de parámetros para generar datos

Para generar los datos, iremos modificando los siguientes parámetros de generación:

- **n_feats**: Número de atributos o dimensiones.
- **compactness_factor**: Cuanto más alto, más compactos serán los clústeres.

- **alpha_n**: Determina los hiperplanos de la cuadrícula. Afecta también a la compacidad de los clústeres. Para más detalles sobre su cálculo, consulte la documentación del generador original [15]. Intentaremos mantenerlo fijo y ajustaremos solo el **compactness_factor** para modificar los clústeres.

- **n_samples_per_period**: Número de instancias por período.

- **n_periods**: Número de períodos.

- **n_old_cluster**: Número de clústeres conocidos.

- **n_new_cluster**: Número de clústeres nuevos.

- **dead_factor**: Puede interpretarse como un factor de envejecimiento. Los valores pueden variar entre 0 y 4; cuanto más alto sea, mayor es la probabilidad de que un clúster sea sustituido en el siguiente período.

- **possible_distributions**: Distribuciones que seguirá el generador para crear los clústeres.

10.4. Clases aisladas y clases compactas

Para los datos con clases aisladas, que son más fáciles de identificar, solo analizaremos un conjunto de datos de este tipo. Sin embargo, los datos con clases compactas son más difíciles de discernir. Utilizaremos una configuración específica de los parámetros de generación para que el conjunto de datos resultante tenga clústeres con sus regiones anómalas solapadas. Para una mejor visualización, en esta sección utilizaremos datos bidimensionales. Generaremos un flujo de datos de dos períodos: uno para el entrenamiento y otro para la predicción. En el primer período, habrá instancias de dos clases, y en el segundo período introduciremos una clase adicional.

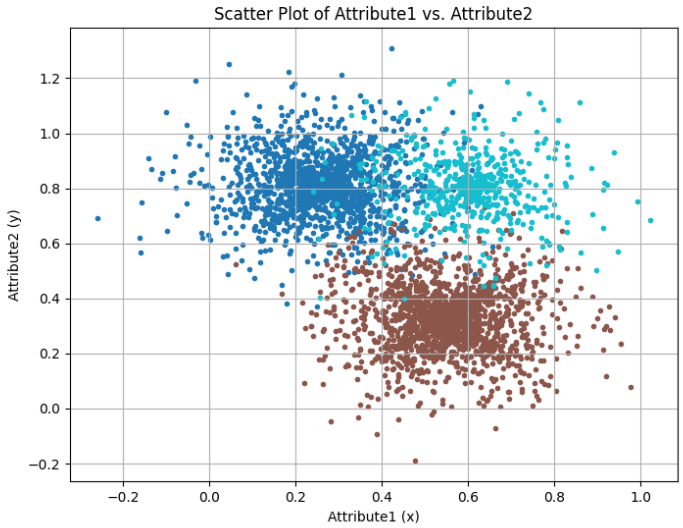

Figura 10.1: DataStream_1. Fuente: generado con DAGADENC [3].

10.4.1. Clases compactas

DataStream_1 10.1. Se usa la siguiente configuración para el generador:

- **n_feats**: 2
- **compactness_factor**: 0.8
- **alpha_n**: 3.5
- **n_samples_per_period**: 2000
- **n_periods**: 3
- **n_old_cluster**: 2
- **n_new_cluster**: 1
- **dead_factor**: 1
- **possible_distributions**: [gaussian]

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.855345	59.314903	0.867	0.74197
2	0.750552	58.648282	0.88	0.77291
3	0.800908	58.591224	0.897	0.81679
4	0.868523	58.323690	0.8875	0.79767
5	0.823753	57.770353	0.8975	0.82051
Average	**0.819816**	**58.52969**	**0.8858**	**0.78997**

Cuadro 10.1: Resultados de las 5 ejecuciones de DataStream_1.

10.4.2. Clases aisladas

DataStream_2 10.5. Utilizaremos la misma configuración que en el apartado anterior, con la única modificación del parámetro `compactness_factor`, el cual se ajustará a 0.1.

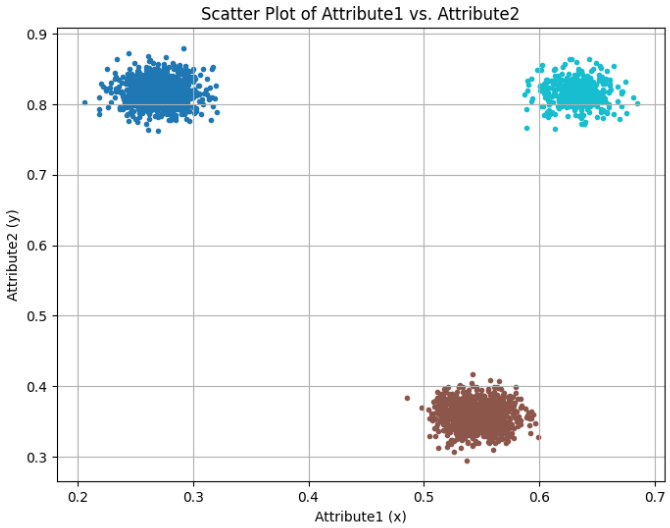

Figura 10.2: DataStream_2. Fuente: generado con DAGADENC [3].

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.734142	53.304567	0.9735	0.95474
2	0.734936	52.938755	0.9695	0.94826
3	0.861183	53.304943	0.9685	0.94666
4	0.861183	53.708499	0.974	0.95556
5	0.732648	54.059019	0.9715	0.95149
Average	**0.78482**	**53.4631566**	**0.9714**	**0.95134**

Cuadro 10.2: Resultados de las 5 ejecuciones de DataStream_2.

10.4.3. Comparativa

En la figura 10.3 podemos ver la comparativa entre DataStream_1 10.4.1 (Compacted) y DataStream_2 10.4.2(Isolated)

Tiempo de entrenamiento

Los tiempos de entrenamiento son bastante similares para ambos conjuntos de datos, lo que indica que la complejidad del modelo en términos de entrenamiento no se ve significativamente afectada por si los clústeres están aislados o compactos.

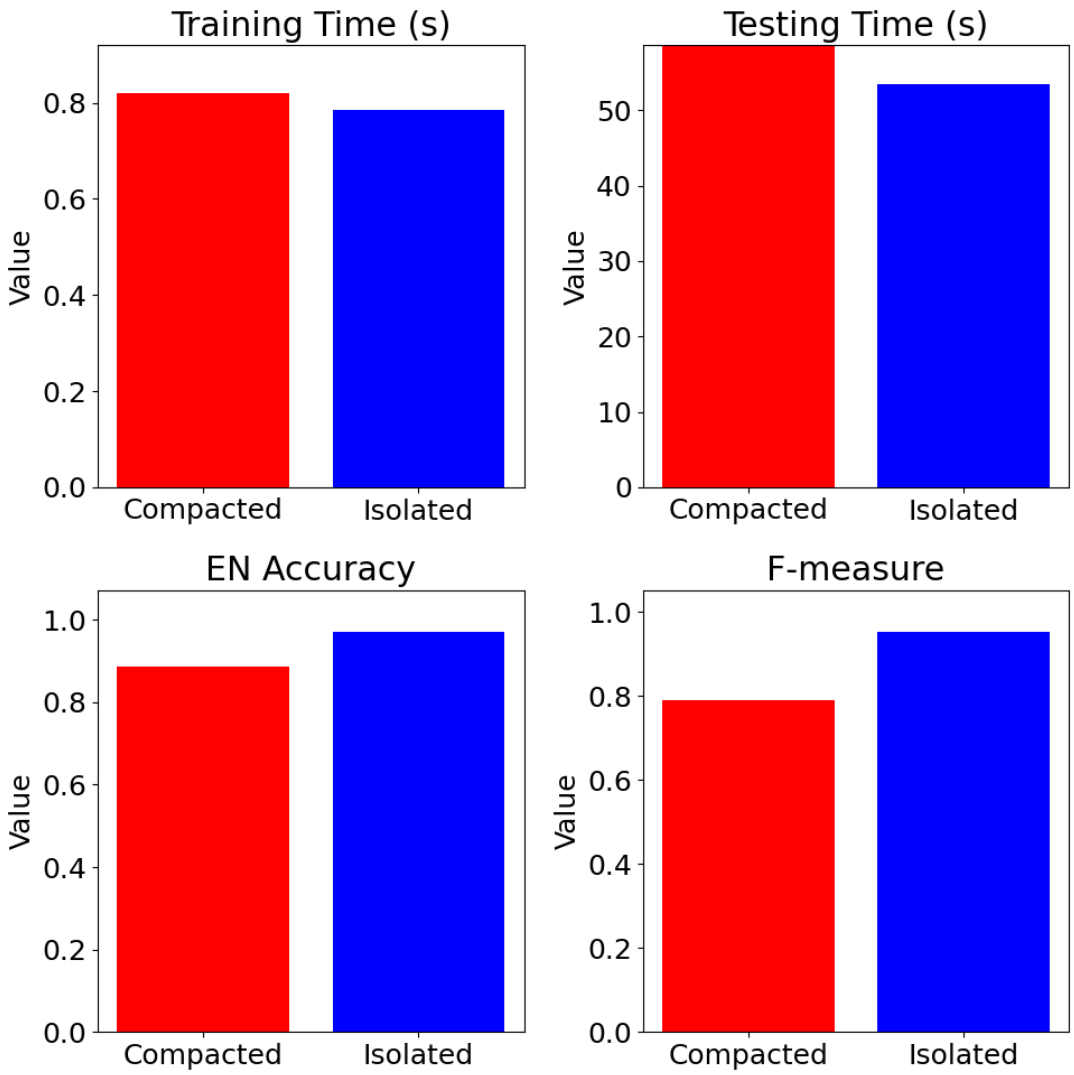

Figura 10.3: Comparativa del promedio (*average*) de las tablas 10.1 y 10.2

Tiempo de prueba o ejecución

El tiempo de prueba es más corto para el conjunto de datos con clústeres aislados. Los clústeres solapados en el otro conjunto de datos pueden hacer la clasificación más desafiante, lo que requiere mayor esfuerzo computacional durante las pruebas. Esto se debe a:

- **Clústeres compactos**: En estas situaciones, las fronteras entre las diferentes clases son menos claras. Esta ambigüedad significa que SENCForest puede necesitar realizar más cálculos para determinar la clase de un punto de datos

específico, ya que podría pertenecer a varias clases simultáneamente.

- **Clústeres aislados**: Cuando los clústeres están claramente separados, la clasificación de un nuevo punto de datos suele ser más sencilla y directa, reduciendo así el esfuerzo computacional necesario.

EN Accuracy

- **Compacted**: La EN Accuracy promedio es de 0.8858.
- **Isorated**: La EN Accuracy promedio es de 0.9714.
- **Análisis**: La EN Accuracy es notablemente más alta (9.11 %) en caso de los aislados. Esto sugiere que el algoritmo SENCForest funciona mejor en escenarios donde los clústeres están bien separados y hay menos ambigüedad en la clasificación de clases nuevas frente a conocidas. Los clústeres solapados probablemente conducen a más clasificaciones erróneas.

F-measure

- **Compacted**: La F-measure promedio es de 0.78997.
- **Isorated**: La F-measure promedio es de 0.95134.
- **Análisis**: Una F-measure significativamente más alta para el caso de Isolated refuerza la conclusión de que el algoritmo SENCForest es más efectivo en condiciones con clústeres aislados y no solapados. En contraste, una F-measure más baja en el conjunto de datos con clústeres solapados señala dificultades para equilibrar precisión y recall. Esto se debe a que la superposición de clases complica la correcta identificación de nuevas clases, aumentando potencialmente los falsos positivos o negativos. Una baja F-measure, en comparación con la EN Accuracy, sugiere una cantidad significativa de falsos positivos o negativos en la detección de clases emergentes. Esta situación podría no impactar de manera tan crítica en la EN Accuracy si la mayoría de las instancias pertenecen a clases ya conocidas y son clasificadas correctamente.

10.5. Análisis del comportamiento con diferentes dimensiones

En esta sección, examinamos los resultados de experimentos llevados a cabo con conjuntos de datos de dimensiones más elevadas, concretamente de 4, 7, 10 y 20 dimensiones. Al igual que en la sección anterior, nuestro enfoque se centrará tanto en datos con clústeres compactos como en aquellos con clústeres aislados. Nuestro objetivo es analizar cómo varía el comportamiento del modelo con el incremento en el número de dimensiones y, finalmente, realizar una comparativa entre los diferentes escenarios.

10.5.1. Clústeres compactos

En este apartado, los resultados se derivan de datos generados siguiendo los parámetros detallados en la sección 10.4.1, utilizando un factor de compactación (*compactness_factor*) fijado en 0.8. Estos conjuntos de datos se caracterizan por presentar clústeres compactos, que pueden exhibir cierto grado de solapamiento entre ellos.

Probaremos cuatro conjuntos de datos:

- *DataStream_3*: datos de 4 dimensiones, clústeres con posibles solapamientos.
- *DataStream_4*: datos de 7 dimensiones, clústeres con posibles solapamientos.
- *DataStream_5*: datos de 10 dimensiones, clústeres con posibles solapamientos.
- *DataStream_6*: datos de 20 dimensiones, clústeres con posibles solapamientos.

En las siguientes tablas, se presentan los resultados promedio de las ejecuciones. Al igual que en la sección anterior, cada conjunto de datos se ejecuta 5 veces y se toma el promedio de los resultados.

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.740251	69.693798	0.7425	0.46968
2	0.763719	66.950026	0.784	0.59198
3	0.820607	67.282176	0.7695	0.5439
4	0.783236	66.468287	0.813	0.66925
5	0.995626	68.375607	0.7535	0.49448
Average	**0.8206878**	**67.7539788**	**0.7725**	**0.553858**

Cuadro 10.3: Resultados de las 5 ejecuciones de DataStream_3 (4 Dimensiones)

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.829736	68.493106	0.6915	0.23634
2	0.826494	68.747341	0.6925	0.2475
3	0.713003	67.875808	0.6965	0.29918
4	0.882638	68.509001	0.689	0.25459
5	0.823686	66.577968	0.6895	0.25245
Average	**0.81511**	**68.04064**	**0.6918**	**0.25801**

Cuadro 10.4: Resultados de las 5 ejecuciones de DataStream_4 (7 Dimensiones)

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.807640	68.971681	0.676	0.27586
2	0.745994	72.227893	0.679	0.26682
3	0.893900	81.547693	0.6795	0.27674
4	0.842376	75.311017	0.685	0.28033
5	1.004235	71.195757	0.6795	0.26887
Average	**0.858829**	**73.850808**	**0.6798**	**0.27372**

Cuadro 10.5: Resultados de las 5 ejecuciones de DataStream_5 (10 Dimensiones)

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.904093	77.135802	0.6465	0.2803
2	0.791030	86.110095	0.6465	0.26539
3	0.760495	75.436411	0.651	0.28059
4	1.016013	79.952400	0.646	0.28807
5	0.800001	85.298670	0.647	0.27033
Average	**0.8543264**	**80.7866756**	**0.6474**	**0.276936**

Cuadro 10.6: Resultados de las 5 ejecuciones de DataStream_6 (20 Dimensiones)

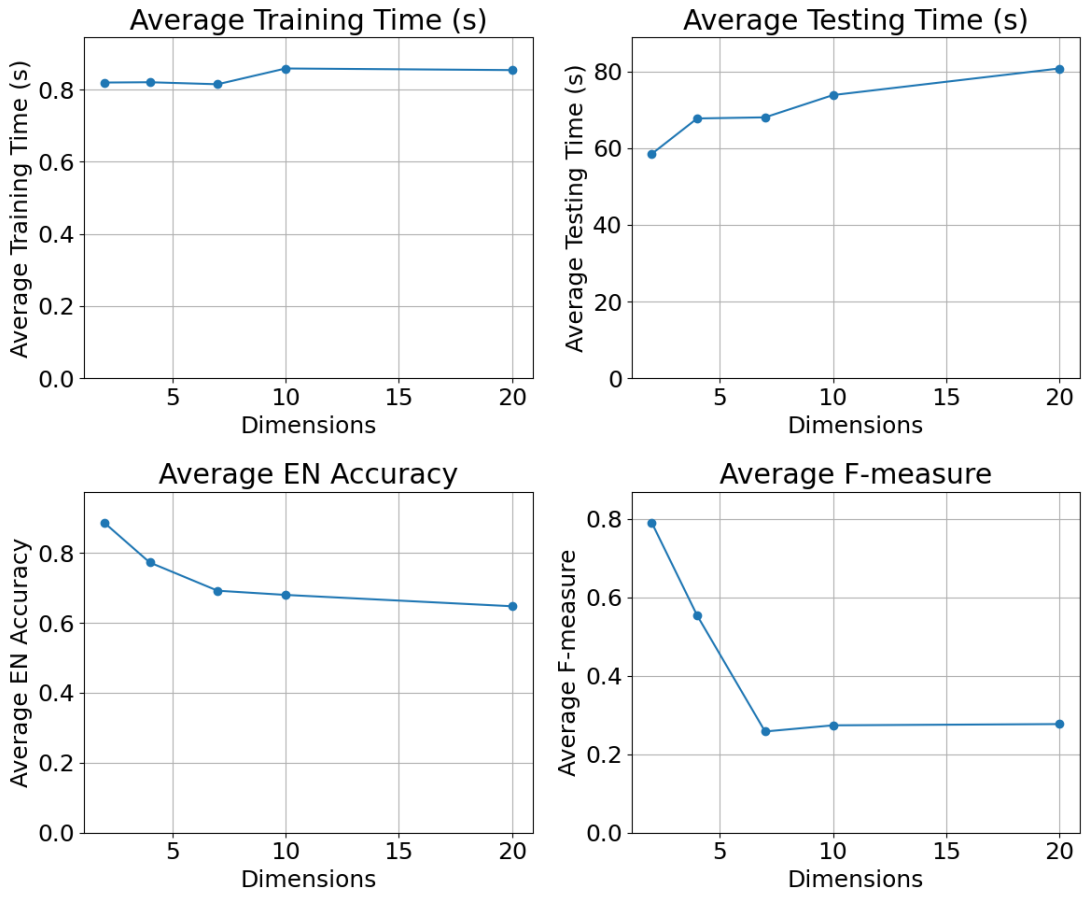

Figura 10.4: Evolución de clústeres con solapamientos. Fuente: generado con DAGADENC [3].

Tiempo de entrenamiento

El tiempo de entrenamiento es bastante rápido, alrededor de 0.8 segundos, y se mantiene relativamente estable con un incremento ligero a medida que aumenta el número de dimensiones. Aunque este incremento es pequeño en comparación con el tiempo de prueba (Testing Time), no es muy representativo para evaluar

el desempeño general del algoritmo.

Tiempo de prueba o ejecución

El tiempo de prueba aumenta con el número de dimensiones. Presenta un incremento lineal, lo que indica que el algoritmo escala de manera razonable con el aumento de las dimensiones de las características

EN Accuracy

La precisión de EN (EN Accuracy) disminuye a medida que aumenta el número de dimensiones. Esto podría sugerir que el modelo enfrenta más dificultades para clasificar con precisión las instancias en espacios de mayor dimensión. Detallando más, esta disminución es notable al pasar de dos a siete dimensiones, donde se observa una caída de aproximadamente 0.2 en la precisión de EN, quedando en alrededor de 0.7 para siete dimensiones. Sin embargo, desde siete hasta 20 dimensiones, la disminución en la precisión de EN es más paulatina; desde siete dimensiones hasta 20, solo ha disminuido en aproximadamente 0.05. Al alcanzar veinte dimensiones, la precisión de EN se mantiene alrededor de 0.65, lo que sugiere que el modelo aún es capaz de ofrecer un nivel razonable de precisión en la clasificación correcta de instancias tanto de clases conocidas como de nuevas clases emergentes.

F-measure

La F-measure muestra un patrón similar a la gráfica de EN Accuracy, con una caída significativa de 2 a 7 dimensiones y una estabilidad de 7 a 20 dimensiones. Sin embargo, la disminución en la F-measure de 2 a 7 dimensiones es bastante pronunciada, bajando de aproximadamente 0.8 a 0.27. Un valor tan bajo de F-measure indica que el modelo enfrenta dificultades para identificar y clasificar correctamente las nuevas clases emergentes. La F-measure es un indicador que combina la precisión (proporción de identificaciones positivas correctas entre todas las identificaciones positivas) y la sensibilidad (recall - proporción de identificaciones positivas correctas entre todas las instancias positivas reales). Un valor bajo sugiere

problemas en una o ambas métricas, lo que implica que el modelo puede estar generando muchos falsos positivos o falsos negativos en la detección de nuevas clases.

Esta situación podría ser el resultado de la complejidad inherente a la separación de clases en espacios de alta dimensión, o de la superposición de características entre clases conocidas y nuevas. Esto sugiere que para dimensiones más altas, el algoritmo necesita una mayor distancia entre los clústeres para lograr una clasificación correcta. Al considerar esto junto con la EN Accuracy, se observa que:

- EN Accuracy relativamente alta: Indica que el modelo es más eficaz en la clasificación general, incluyendo tanto las clases conocidas como las nuevas. Esto puede señalar que, a pesar de los desafíos con las nuevas clases, el modelo aún logra un rendimiento razonable en la clasificación de instancias en general.

- Es posible que el modelo demuestre un mejor rendimiento en identificar y clasificar clases conocidas en comparación con las nuevas clases emergentes.

10.5.2. Clústeres aislados

En este apartado, los resultados se obtienen de datos generados siguiendo los parámetros especificados en la sección 10.4.1, con un factor de compactación (*compactness_factor*) establecido en 0.1. Estos conjuntos de datos se caracterizan por tener clústeres aislados, con una probabilidad muy baja de solapamiento entre ellos.

Probaremos cuatro conjuntos de datos:

- *DataStream_7*: datos de 4 dimensiones, clústeres aislados.

- *DataStream_8*: datos de 7 dimensiones, clústeres aislados.

- *DataStream_9*: datos de 10 dimensiones, clústeres aislados.

- *DataStream_10*: datos de 20 dimensiones, clústeres aislados.

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.734818	57.754230	0.9255	0.8824
2	0.930024	58.405615	0.926	0.8831
3	0.717315	60.099811	0.936	0.89727
4	1.003912	59.408008	0.929	0.8873
5	0.887398	59.706208	0.933	0.89297
Average	**0.8546934**	**59.0747744**	**0.9299**	**0.888608**

Cuadro 10.7: Resultados de las 5 ejecuciones de DataStream_7 (4 Dimensiones)

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.798506	61.191144	0.922	0.87755
2	1.070560	60.452981	0.9235	0.87962
3	0.870854	60.934958	0.9205	0.87549
4	0.840719	60.825312	0.925	0.8817
5	0.887398	61.357803	0.921	0.87618
Average	**0.8936074**	**60.9524396**	**0.9224**	**0.878108**

Cuadro 10.8: Resultados de las 5 ejecuciones de DataStream_8 (7 Dimensiones)

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	0.939317	62.142429	0.9185	0.87276
2	1.046178	62.246937	0.9185	0.87276
3	0.898972	63.952536	0.9175	0.8714
4	0.966925	64.107254	0.9205	0.87549
5	1.040439	61.804353	0.915	0.86801
Average	**0.9783662**	**62.8507018**	**0.918**	**0.872084**

Cuadro 10.9: Resultados de las 5 ejecuciones de DataStream_9 (10 Dimensiones)

No.	Training Time (s)	Testing Time (s)	EN Accuracy	F-measure
1	1.080842	68.525927	0.8905	0.8362
2	0.762421	63.409270	0.8895	0.83495
3	0.944821	64.733788	0.887	0.83185
4	1.035487	64.239102	0.8815	0.82509
5	1.006549	64.339745	0.879	0.82206
Average	**0.966024**	**65.0495664**	**0.8855**	**0.83003**

Cuadro 10.10: Resultados de los 5 ejecuciones de DataStream_10 (20 Dimensiones)

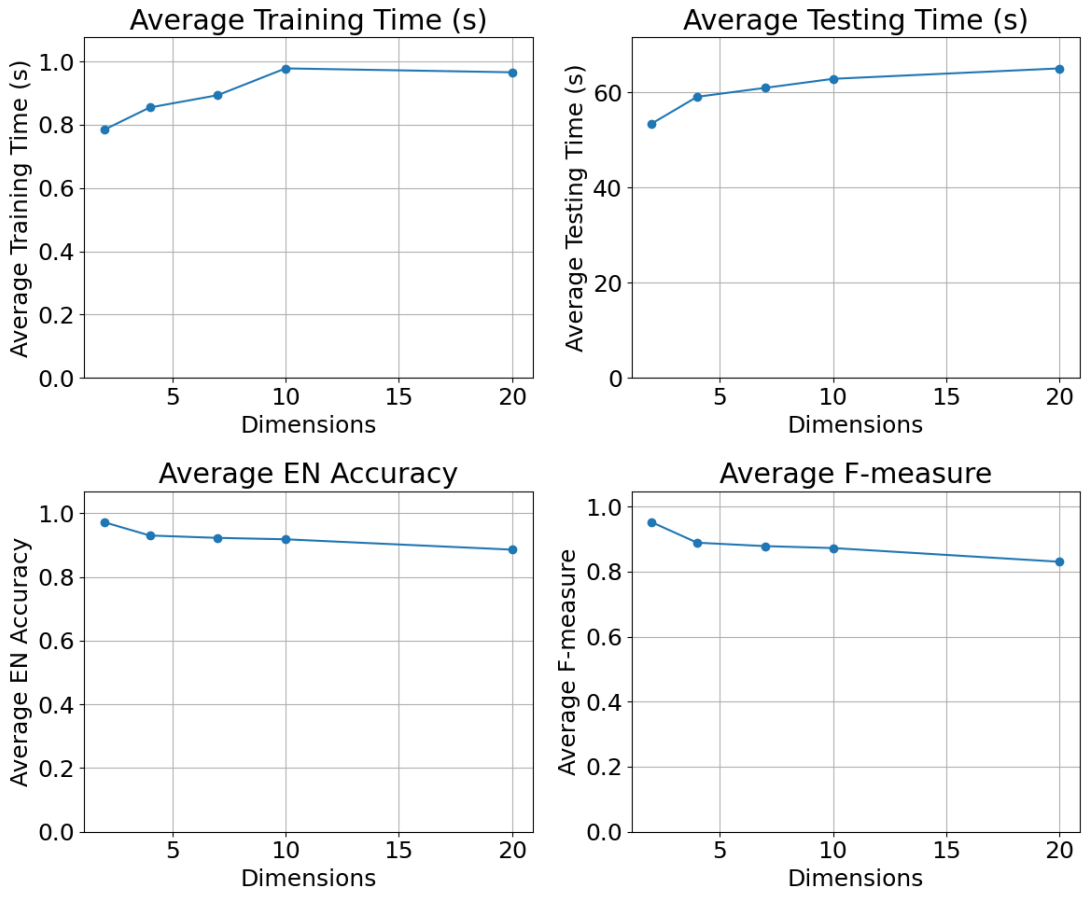

Figura 10.5: Evolución de clústeres aislados. Fuente: generado con DAGADENC [3].

Analizaremos cómo el modelo SENCForest se adapta a cambios en la dimensionalidad de los datos en un entorno más controlado, donde los clústeres están aislados y no se superponen. Observamos que las gráficas del tiempo de entrenamiento y de prueba se comportan de manera similar al caso anterior con clústeres solapados. Sin embargo, una diferencia notable es que el tiempo de prueba es mejor en estas

condiciones que en el caso anterior, lo que sugiere que el algoritmo es más eficaz en entornos con clústeres aislados. Lo más interesante son las gráficas de EN Accuracy y F-measure:

EN Accuracy

La precisión de EN disminuye ligeramente con el aumento de las dimensiones, pero se mantiene en niveles relativamente altos, incluso en espacios de 20 dimensiones, donde la precisión se mantiene en torno al 0.88. Comparando con el apartado anterior, observamos que la disminución de la precisión en este escenario es mucho más gradual y menos pronunciada.

Esto indica que el modelo SENCForest es bastante robusto en la clasificación correcta de instancias en contextos donde los clústeres están aislados, incluso en espacios de alta dimensión. La capacidad del modelo para mantener una alta precisión de en estas condiciones sugiere una eficacia constante en la identificación tanto de clases conocidas como de nuevas clases emergentes, incluso cuando se aumenta la complejidad del espacio de características.

F-measure

Aunque la F-measure disminuye ligeramente con el aumento de las dimensiones, se mantiene en un nivel relativamente alto, alcanzando un valor de aproximadamente 0.83 para datos de 20 dimensiones. En comparación con el caso anterior, donde para 20 dimensiones la F-measure era de solo alrededor de 0.27, esta mejora es significativa.

Este comportamiento sugiere que el modelo SENCForest es efectivo en mantener un buen equilibrio entre precisión y sensibilidad (recall) en condiciones donde los clústeres aislados, especialmente en la detección de nuevas clases, incluso a medida que aumenta la complejidad dimensional de los datos. La alta F-measure en este escenario de clústeres aislados indica que el modelo logra identificar y clasificar correctamente las nuevas clases emergentes sin generar un número excesivo de falsos positivos o falsos negativos. Esto refleja la eficacia y la robustez del modelo en entornos de datos con clústeres claramente definidos y separados.

11 Sostenibilidad

Analizaremos la sostenibilidad basándonos en la matriz de sostenibilidad.

11.1. Dimensión ambiental

11.1.1. Proyecto Puesto en Producción (PPP)

El impacto ambiental directo de este proyecto, enfocado en la investigación académica, es notablemente bajo. La naturaleza digital del proyecto implica que no se realiza producción física ni se consume una cantidad significativa de recursos naturales. El proyecto se ha desarrollado íntegramente en un ordenador personal, cuyo consumo promedio es de aproximadamente 0.22 kWh.

Para minimizar aún más el impacto ambiental, se han procedido a la reutilización de herramientas, como el empleo de software de código abierto y la utilización de algoritmos ya existentes, es un claro ejemplo de esta práctica. Esta estrategia no solo es sostenible, sino que también contribuye a la eficiencia general del proyecto.

Aunque el proyecto se ha diseñado para ser lo más eficiente posible en términos de recursos, siempre existe margen para la optimización. Por ejemplo, mejorar la eficiencia del código puede contribuir a reducir el consumo de energía durante la computación.

11.1.2. Vida útil

Estado del arte y mejoras ambientales de la solución propuesta

El problema abordado en este proyecto, la detección de anomalías en flujos de datos, es actualmente una cuestión de interés en el campo de la ciencia de datos y la inteligencia artificial. Las soluciones existentes suelen requerir una infraestructura computacional considerable, lo que conlleva un uso energético significativo. Nuestra solución, al ser un software basado en algoritmos eficientes y optimizados, busca mejorar ambientalmente estas soluciones existentes.

Recursos estimados durante la vida útil y su impacto ambiental

Durante la vida útil del proyecto, se estima que el principal recurso utilizado con impacto ambiental será la energía eléctrica, destinada principalmente al funcionamiento de los equipos informáticos donde se ejecutará el software desarrollado. Este consumo energético está asociado a ordenadores personales, que generalmente tienen un impacto ambiental más bajo en comparación con grandes servidores o centros de datos.

Reducción del uso de otros recursos y huella ecológica

El proyecto podría contribuir a reducir el uso de recursos en empresas o contextos donde se implemente. Por ejemplo, al optimizar la detección de anomalías o la detección de novedades, podría disminuir la necesidad de procesamiento de datos en servidores, resultando en un menor consumo de energía.

11.1.3. Riesgos en la fase inicial

1. **Uso intensivo de recursos computacionales**: Una de las situaciones que podría aumentar la huella ecológica del proyecto es un uso intensivo y prolongado de recursos computacionales. Si el algoritmo requiere un alto grado de potencia de procesamiento o se ejecuta continuamente en múltiples máquinas, esto podría incrementar el consumo de energía.

2. **Almacenamiento y Procesamiento de Datos**: En la medida en que el proyecto se amplíe o se utilice para manejar grandes volúmenes de datos, podría requerir una mayor capacidad de almacenamiento y procesamiento.

11.2. Dimensión económica

11.2.1. Proyecto Puesto en Producción (PPP)

Estimación y cuantificación del coste

El coste de la realización del proyecto ha sido cuidadosamente estimado y cuantificado, teniendo en cuenta tanto los recursos humanos como los materiales. El proyecto, al ser principalmente de desarrollo e investigación de software, ha implicado costes asociados al tiempo de trabajo del personal (estudiante y director del proyecto) y el uso de equipos informáticos. Se estimaron y documentaron los costes de personal según las horas invertidas, y los costes materiales se basaron en el uso de un ordenador personal.

Decisiones para reducir el coste

Para minimizar los costes del proyecto, se implementaron varias decisiones:

1. **Uso de software de código abierto**: Se optó por el uso de software libre para diferentes aspectos del proyecto. Esta elección no solo evitó costes adicionales, sino que también fomentó la adaptabilidad y la flexibilidad en el desarrollo.

2. **Reutilización de algoritmos existentes**: En lugar de desarrollar algoritmos desde cero, se prefirió adaptar y optimizar algoritmos ya existentes. Esta estrategia redujo significativamente el tiempo y los recursos necesarios para el desarrollo y la investigación.

3. **Aprovechamiento de ofertas para estudiantes**: Al ser un proyecto universitario, se pudieron utilizar varias herramientas de software especializado de forma gratuita, como PyCharm. Estas ofertas para estudiantes permitieron acceder a recursos de alta calidad sin aumentar los gastos.

Ajuste del coste y lecciones aprendidas

El coste final del proyecto se ajustó de manera eficiente al presupuesto previsto. Las pequeñas diferencias surgidas se debieron principalmente a ajustes en la fase de exploración y aprendizaje, donde se dedicó más tiempo de lo inicialmente estimado. Estas diferencias se justificaron por la necesidad de una comprensión más profunda de los algoritmos y herramientas seleccionadas. Como lección aprendida, se destaca la importancia de un margen de flexibilidad en la planificación del tiempo y los recursos para acomodar los desafíos inesperados, típicos de un proyecto de investigación y desarrollo.

11.2.2. Vida útil

Estado actual del problema:

1. **Estado Actual**: Actualmente, la detección de anomalías y la detección de novedades en flujos de datos se realiza mediante una variedad de algoritmos y sistemas, algunos de los cuales pueden ser complejos y costosos de implementar y mantener.

2. **Mejoras económicas de la solución**: La solución propuesta en este proyecto se basa en la optimización y adaptación de algoritmos existentes, lo que reduce la necesidad de desarrollo desde cero. Al usar software de código abierto y algoritmos ya probados, se reduce el coste de implementación y mantenimiento, mejorando así la viabilidad económica en comparación con las soluciones actuales.

Coste estimado durante la vida útil:

1. **Coste de operación**: El principal coste durante la vida útil del proyecto será el consumo eléctrico para operar los equipos informáticos.

2. **Reducción de costes y viabilidad**: Dado el enfoque en la eficiencia y la reutilización de recursos, los costes operativos ya son bajos. Sin embargo,

siempre existe la posibilidad de optimización adicional, especialmente en el código, para reducir aún más el consumo de energía y los costes asociados.

3. **Costes de Ajustes/Actualizaciones/Reparaciones**: Se prevé que el mantenimiento y las actualizaciones del software tendrán un coste mínimo, ya que el proyecto se basa en plataformas de código abierto y en la colaboración comunitaria.

11.2.3. Riesgos en la fase inicial

Dependencia de software de terceros

Si el proyecto depende de software o herramientas de terceros, cualquier cambio en la licencia, coste o disponibilidad de estos recursos podría impactar negativamente en la viabilidad económica del proyecto.

Necesidad de estar especializado

La implementación y el mantenimiento continuo del proyecto pueden requerir habilidades y conocimientos técnicos especializados, especialmente en el ámbito de la detección de anomalías y el procesamiento de *data streams*. Aunque actualmente se ha manejado esta necesidad con los recursos existentes, futuras expansiones o actualizaciones podrían requerir inversión en formación o incluso la contratación de expertos. Dado que algunos de estos conocimientos pueden estar sujetos a cursos o recursos de acceso restringido, podría ser necesario considerar costos adicionales para la capacitación y actualización de habilidades.

11.3. Dimensión social

11.3.1. Proyecto Puesto en Producción (PPP)

Aportes a nivel personal:

1. **Desarrollo profesional y técnico**: El proyecto ha requerido una extensa fase de aprendizaje de habilidades técnicas, incluyendo el manejo de herramientas de análisis de datos como Python y sus librerías, así como una introducción

al lenguaje de programación Matlab. Además, ha permitido adentrarse en el campo de detección de anomalías y novedades en flujos de datos, contribuyendo al crecimiento profesional y técnico en el ámbito de la ingeniería informática.

2. **Aprendizaje continuo y adaptabilidad**: La naturaleza investigativa del proyecto ha impulsado un aprendizaje autónomo significativo y la capacidad de adaptarse a nuevos retos y conocimientos. La fase más exigente, la exploración de algoritmos, representó un desafío personal considerable debido a la necesidad de encontrar una solución adecuada dentro de un amplio espectro de opciones disponibles.

3. **Superación de retos y crecimiento personal**: El proyecto ha requerido enfrentar y superar retos sustanciales, especialmente durante la fase de exploración de algoritmos. Esta etapa ha sido de suma importancia para el desarrollo personal, la satisfacción al haber encontrado una solución apropiada a los requisitos establecidos en un mar de conocimientos, para mi, es una felicidad alto nivel.

4. **Colaboración y contribución a la comunidad**: El proyecto ha fomentado un sentido de colaboración y contribución a la comunidad de software al adaptar y emplear herramientas y algoritmos de código abierto. Esta experiencia ha reforzado la comprensión de la importancia de compartir conocimientos y recursos tanto en el entorno académico como en el profesional. Estoy muy agradecido a los investigadores que compartieron los resultados de su trabajo, lo que ha sido esencial para el desarrollo de este proyecto.

5. **Interés**: El desarrollo de este proyecto ha despertado mi interés en áreas relacionadas con el análisis de datos. La exploración de conceptos complejos y la aplicación de técnicas avanzadas de detección de anomalías y análisis de *data streams* han ampliado significativamente mi curiosidad y entusiasmo por estas áreas. La experiencia adquirida durante el proyecto ha abierto nuevas vías de exploración y aprendizaje continuo, motivándome a seguir profundizando

en estos campos y explorando oportunidades para aplicar y expandir estos conocimientos en futuros proyectos y contextos profesionales.

11.3.2. Vida útil

1. **Necesidad real del proyecto**: Existe una necesidad real y creciente en el campo de la informática y el análisis de datos, particularmente en la detección de anomalías o novedades en flujos de datos. Este proyecto aborda esta necesidad al proporcionar herramientas y enfoques mejorados para analizar *data streams* en busca de comportamientos inusuales o nuevos patrones, lo que es vital en numerosos sectores, como la seguridad informática y la detección de fraudes financieros.

2. **Beneficiarios del proyecto**: Los principales beneficiarios del proyecto son investigadores, estudiantes y profesionales en campos relacionados con el análisis de datos y la inteligencia artificial. Además, empresas y organizaciones que trabajan con grandes volúmenes de datos en tiempo real podrían encontrar en este proyecto una herramienta valiosa para mejorar sus procesos de toma de decisiones y eficiencia operativa.

3. **Posibles efectos negativos**: No se identifican colectivos que puedan verse perjudicados directamente por el proyecto. Sin embargo, es esencial recordar a los usuarios del proyecto o que usaran frutos de este proyecto, sean conscientes de estas implicaciones éticas y utilicen las herramientas de manera responsable.

4. **Solución al problema planteado**: El proyecto responde efectivamente al problema inicialmente planteado de mejorar la detección y análisis de anomalías en flujos de datos. Al integrar y optimizar algoritmos existentes, y al desarrollar un generador de datos sintéticos adaptado, el proyecto no solo cumple con su propósito inicial, sino que también proporciona una base sólida para futuras investigaciones y aplicaciones en este campo. La disponibilidad de este recurso como una biblioteca de software de código abierto amplía su accesibilidad y potencial de impacto en la comunidad académica y profesional.

12
Futuros trabajos

Este trabajo es una introducción al campo de estudio de la detección de anomalías y novedades en *data streams*. A lo largo del desarrollo del proyecto, hemos identificado diversas posibilidades para su continuación o expansión. A continuación, presentamos algunas ideas que podrían ser de utilidad para futuros trabajos. Cabe destacar que estas son solo propuestas conceptuales; no se ha realizado un análisis de viabilidad y, por tanto, no se puede asegurar su factibilidad.

12.1. El generador de datos

El generador de datos presentado en este trabajo, DAGADENC, presenta varios aspectos susceptibles de mejora. Detallamos algunos a continuación.

1. **Estructura de periodos más flexible**

   El generador actual, estructurado en períodos predefinidos, puede limitar su aplicabilidad en contextos más realistas. En situaciones reales, la aparición y desaparición de clústeres no son predecibles y no ocurren en momentos específicos. Una mejora significativa sería desarrollar un generador que simule este comportamiento impredecible de los clústeres.

   Propuesta: El desarrollo de un generador que determine en cada instante si debe crearse o destruirse un clúster. Esta decisión podría basarse en funciones

de distribución probabilística. Por ejemplo, utilizando la distribución de Poisson para determinar la necesidad de insertar un nuevo clúster, o la distribución Exponencial para gestionar su desaparición, siempre intentando mantener un número constante de clústeres, según lo definido por el usuario.

Potencial de la Idea: Esta propuesta abriría nuevas posibilidades para simular escenarios más cercanos a datos reales, donde los cambios en los clústeres son menos predecibles y más dinámicos. Sería un paso hacia adelante en la simulación de *data streams* más realista, proporcionando una herramienta más robusta y adaptable para futuras investigaciones en la detección de anomalías y novedades.

Consideraciones Adicionales: Aunque esta es una idea conceptual, sería necesario realizar un análisis más profundo para determinar su viabilidad técnica y práctica. La implementación de esta propuesta requeriría un entendimiento profundo de las distribuciones probabilísticas y su aplicación en la generación de datos.

2. **Generación en tiempo real**

   El generador de datos actual sigue una estructura predeterminada, en la que el orden de generación de las instancias de cada clúster se establece antes de generar las instancias mismas. Aunque esto facilita la manipulación del *data stream* completo, presenta limitaciones para la generación de datos de gran magnitud. Por ejemplo, durante la generación del *data stream*, pueden surgir situaciones donde algunas instancias resulten innecesarias o donde se produzcan interrupciones, especialmente si el volumen de datos es muy grande.

   Propuesta: La solución propuesta en el punto anterior, de realizar decisiones en tiempo real sobre el incremento o decremento de los clústeres, también aplicaría aquí. El concepto es mantener un número constante de clúster a nivel global, pero con decisiones tomadas en cada instante sobre qué clústeres generar. Este enfoque permitiría una generación de datos más dinámica y

adaptable, donde en cada instante se escogerían los clústeres *vivos* de un *estanque* de posibles clústeres, basándose en una distribución de probabilidad.

Beneficios Potenciales: Este enfoque mejoraría la capacidad del generador para simular escenarios más realistas y dinámicos, donde los clústeres varían de manera más natural y espontánea. Sería particularmente útil para investigaciones que requieran *data streams* que simulen condiciones de mundo real, como en aplicaciones de IoT o monitoreo en tiempo real.

Para abordar los puntos planteados sobre la generación de *data streams* y la gestión de clústeres, la implementación de un mecanismo basado en el algoritmo de reemplazo LRU (Least Recently Used) parece ser una solución prometedora. A continuación, se detalla cómo podría adaptarse este algoritmo al contexto del proyecto:

1. **Selección de clústeres**: En cada instante, se deberá seleccionar un clúster del conjunto de clústeres activos para generar las instancias. Este proceso se puede realizar tratando el conjunto de clústeres como una caché, donde cada clúster es un elemento en esta caché.

2. **Mantenimiento de un número constante de clústeres**: Se puede establecer un tamaño fijo para el *estanque* de clústeres, correspondiente al número de clústeres activos que se deben mantener a lo largo de todo el *data stream*. Esto ayuda a gestionar el conjunto de clústeres de manera eficiente y dinámica.

3. **Gestión de la Aparición y Desaparición de Clústeres**: Dado que los clústeres no aparecen o desaparecen con alta frecuencia, se puede utilizar el algoritmo LRU para determinar cuál clúster debe ser reemplazado cuando sea necesario insertar un nuevo clúster y el *estanque* esté lleno. El clúster menos recientemente utilizado sería el candidato para ser sustituido.

Algoritmo LRU en el generador:

- El *estanque* de clústeres se trataría como una caché de tamaño fijo, donde el tamaño está determinado por el número máximo de clústeres activos permitidos en cualquier momento.

- Cada vez que se genera una instancia de un clúster, este se marca como *recientemente utilizado* en la caché.
- Si se necesita insertar un nuevo clúster y la caché está llena, el algoritmo LRU identificaría el clúster menos recientemente utilizado para ser reemplazado.

Al igual que en los casos anteriores, esta propuesta es una idea conceptual y no se ha analizado su viabilidad técnica en profundidad.

12.2. Profundizar en el campo de SENC

A lo largo de este proyecto, hemos examinado un algoritmo específico, SENCForest, que aborda el desafío del Streaming con Emerging New Classes (SENC), aunque hemos observado ciertas limitaciones en su aplicación a *data streams* reales. Nuestra elección de SENCForest se debió a su simplicidad relativa en el contexto de la literatura existente. Sin embargo, existen otros trabajos en este campo que abordan el problema desde diferentes perspectivas y que podrían ofrecer enfoques más avanzados o efectivos. A continuación, proporcionamos referencias a algunos de estos trabajos, los cuales no hemos explorado en detalle, pero representan direcciones interesantes para futuras investigaciones:

1. **Multi-Classification Generative Adversarial Network for Streaming Data with Emerging New Classes: Method and its Application to Condition Monitoring** [12]: Este artículo presenta un enfoque basado en redes generativas adversarias (GAN) para clasificación en *data streams*.
2. **Nearest Neighbor Ensembles: An Effective Method for Difficult Problems in Streaming Classification with Emerging New Classes** [17].

13
Conclusión

Este Trabajo de Fin de Grado ha representado un viaje de exploración en el campo de la detección de anomalías en *data streams*. A lo largo de este proyecto, hemos abordado el desafío de identificar y aplicar algoritmos eficientes para la detección de novedades y anomalías en flujos de datos continuos, una tarea importante en el mundo de la informática y el análisis de datos.

Nuestros logros y contribuciones principales en el campo incluyen:

- La implementación de DAGADENC, un generador de datos sintéticos, adaptado para la detección de novedades en *data streams*. Este generador, una versión modificada del utilizado por los autores del algoritmo SDO, ha demostrado ser una herramienta eficaz en la generación de conjuntos de datos experimentales adecuados para algoritmos como SENCForest, enfocados en la detección de novedades.

- La adaptación y análisis del algoritmo SENCForest, lo que ha facilitado una comprensión más profunda de sus fortalezas y limitaciones en variados contextos y condiciones de datos. Además, hemos contribuido en adaptar el código de SENCForest para que trabaje con los datos generados por DAGADENC, así como implementando y evaluando métricas de eficacia y eficiencia, como la EN Accuracy y la F-measure.

- La publicación de todo el material, incluyendo el código fuente y la documentación en un repositorio de GitHub, lo que facilita el acceso y uso por parte de otros estudiantes e investigadores interesados en este campo.

Para llegar a estos logros, nos hemos enfrentado a desafíos difíciles y variados. Entre los desafíos más notables se encuentran:

- **Falta de conocimiento inicial sobre el campo de estudio**: La necesidad de una extensa fase de aprendizaje autónomo fue uno de los primeros y más grandes retos. Esta fase de exploración inicial, caracterizada por su incertidumbre, resultó ser especialmente desafiante. Como área de estudio desconocida y con un tiempo limitado, encontrar una dirección clara fue un proceso desafiante.

- **Rápida adaptación a herramientas existentes**: El proyecto requería el uso y comprensión de diversas herramientas y lenguajes de programación. Algunos ejemplos incluyen LaTeX, Python y sus librerías, y especialmente Matlab. Dado que el código de SENCForest estaba escrito en Matlab y mi experiencia previa en análisis de datos con este lenguaje era limitada, la adaptación fue un proceso exigente. Esta limitación en el manejo de Matlab impidió realizar un análisis profundo del funcionamiento interno de SENCForest, lo cual afectó la posibilidad de experimentar con la eficiencia de actualización del algoritmo.

Un último punto a destacar es que el proyecto no solo ha contribuido a la comunidad académica a través de sus resultados, sino que también ha abierto nuevas vías para futuras investigaciones y desarrollos en el campo. Las herramientas y metodologías desarrolladas en este trabajo pueden servir como base para investigaciones más avanzadas o aplicaciones prácticas en la detección de anomalías en *data streams*.

En conclusión, este Trabajo de Fin de Grado ha cumplido con sus objetivos iniciales, llevándome a través de un valioso viaje de aprendizaje en el campo de la detección de anomalías en *data streams*. Por mi parte, este proceso ha despertado mi interés en avanzar y profundizar aún más en este campo. Por último, quiero expresar

mi profunda gratitud a los autores que compartieron sus investigaciones, facilitando así el desarrollo de este trabajo. Un especial agradecimiento al director de este proyecto, Conrado Martínez Parra, por su apoyo y orientación. Espero sinceramente que este proyecto pueda servir como recurso útil para futuras investigaciones en este campo.

Referencias

[1] Lukas Ruff et al. «A Unifying Review of Deep and Shallow Anomaly Detection». En: *Proceedings of the IEEE* 109.5 (mayo de 2021), págs. 756-795. URL: http://dx.doi.org/10.1109/JPROC.2021.3052449.

[2] *Mdcgenpy*. URL: https://github.com/CN-TU/mdcgenpy/tree/master.

[3] *Repositorio del TFG*. URL: https://github.com/IDBIDO/TFG.

[4] *Trello*. URL: https://trello.com.

[5] *Overleaft*. URL: https://www.overleaf.com/.

[6] *talent.com*. URL: https://es.talent.com/salary?job=jefe+de+proyecto+software.

[7] *practicas FIB*. URL: https://www.fib.upc.edu/es/empresa/practicas-en-empresa.

[8] Azzedine Boukerche, Lining Zheng y Omar Alfandi. «Outlier Detection: Methods, Models, and Classification». En: *ACM Comput. Surv.* 53.3 (jun. de 2020). URL: https://doi.org/10.1145/3381028.

[9] Félix Iglesias Vázquez, Tanja Zseby y Arthur Zimek. «Outlier Detection Based on Low Density Models». En: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. 2018, págs. 970-979.

[10] Félix Iglesias et al. «SDOclust: Clustering withnbsp;Sparse Data Observers». En: *Similarity Search and Applications: 16th International Conference, SISAP 2023, A Coruña, Spain, October 9–11, 2023, Proceedings*. Coruna, Spain: Springer-Verlag, 2023, págs. 185-199. URL: https://doi.org/10.1007/978-3-031-46994-7_16.

[11] Xin Mu, Kai Ming Ting y Zhi-Hua Zhou. *Classification under Streaming Emerging New Classes: A Solution using Completely Random Trees*. 2016. arXiv: 1605.09131 [cs.LG].

[12] Yu Wang y Alexey Vinogradov. «Multi-classification generative adversarial network for streaming data with emerging new classes: method and its application to condition monitoring». En: (oct. de 2022). URL: http://dx.doi.org/10.36227/techrxiv.21176956.v1.

[13] *Wikipedia. Isolation forest*. URL: https://en.wikipedia.org/wiki/Isolation_forest.

[14] Félix Iglesias et al. «MDCStream: Stream Data Generator for Testing Analysis Algorithms». En: *Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools*. VALUETOOLS '20. Tsukuba, Japan: Association for Computing Machinery, 2020, págs. 56-63. URL: https://doi.org/10.1145/3388831.3388832.

[15] *MDCGenpy documentation*. URL: https://mdcgenpy.readthedocs.io/en/latest/index.html.

[16] *Repositorio de SENCForest*. URL: https://github.com/Orzza/SENCForest.

[17] Xin-Qiang Cai et al. «Nearest Neighbor Ensembles: An Effective Method for Difficult Problems in Streaming Classification with Emerging New Classes». En: *2019 IEEE International Conference on Data Mining (ICDM)*. 2019, págs. 970-975.