UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Implementación del algoritmo TRA-CLUS Mejorado para el análisis de tráfico

Presentado por Álvaro González Delgado
en Universidad de Burgos — 7 de enero de 2025
Tutor: Bruno Baruque Zanon y Hector Cogollos Adrian

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Álvaro González Delgado, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 7 de enero de 2025

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

El presente proyecto tiene como objetivo el desarrollo de una aplicación web interactiva para la visualización y análisis de trayectorias mediante la implementación del algoritmo de agrupamiento de trayectorias TRACLUS y el estudio de sus variantes. La solución integra herramientas de procesamiento de datos y visualización para facilitar la interpretación de patrones espaciales en trayectorias, con énfasis en eficiencia y aplicabilidad en entornos de Big Data. Se emplearon metodologías ágiles y tecnologías modernas como Dash y scikit-learn para garantizar un diseño modular y adaptable.

Descriptores

TRACLUS, análisis de trayectorias, agrupamiento, visualización de datos, Dash, scikit-learn, Big Data.

Abstract

This project aims to develop an interactive web application for the visualization and analysis of trajectories through the implementation of the TRACLUS trajectory clustering algorithm and the study of its variants. The solution integrates data processing and visualization tools to facilitate the interpretation of spatial patterns in trajectories, focusing on efficiency and applicability in Big Data environments. Agile methodologies and modern technologies such as Dash and scikit-learn were employed to ensure a modular and adaptable design.

Keywords

TRACLUS, trajectory analysis, clustering, data visualization, Dash, scikit-learn, Big Data.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

El presente trabajo aborda el desarrollo e implementación de un sistema orientado al análisis de grandes volúmenes de datos geoespaciales mediante técnicas de agrupamiento o *clustering*. En concreto, se ha centrado en el algoritmo TRA-CLUS, diseñado para segmentar y agrupar trayectorias GPS, y en la creación de una aplicación web interactiva que facilite la interpretación y comparación de los resultados obtenidos con diferentes algoritmos de clustering.

El análisis de trayectorias es una tarea fundamental en campos como la movilidad urbana, la gestión del tráfico y la planificación territorial. Sin embargo, la complejidad y el volumen de datos asociados a estas áreas hacen necesario el uso de algoritmos eficientes y herramientas visuales que permitan extraer información útil de manera intuitiva. Este proyecto tiene como objetivo ofrecer una solución integral que combine una implementación optimizada del algoritmo TRA-CLUS con una interfaz visual de fácil uso.

La estructura de esta memoria está diseñada para presentar de manera clara y ordenada el desarrollo del proyecto:

- **Capítulo 1 - Introducción**: Introduce el contexto general del proyecto, sus motivaciones y la organización de esta memoria.
- **Capítulo 2 - Objetivos del Proyecto**: Detalla los objetivos generales, técnicos y personales que han guiado el desarrollo del trabajo.
- **Capítulo 3 - Conceptos Teóricos**: Expone los fundamentos teóricos relacionados con el análisis de trayectorias, el clustering y las técnicas utilizadas en el desarrollo del proyecto.

- **Capítulo 4 - Técnicas y Herramientas**: Describe las tecnologías y herramientas empleadas, justificando su elección y explicando su aplicación dentro del proyecto.
- **Capítulo 5 - Aspectos Relevantes del Proyecto**: Presenta las decisiones clave tomadas durante el desarrollo, los desafíos enfrentados y las soluciones implementadas.
- **Capítulo 6 - Trabajos Relacionados**: Revisa proyectos y estudios previos que guardan relación con este trabajo, destacando sus aportaciones y diferencias con el presente proyecto.
- **Capítulo 7 - Conclusiones y Líneas de Trabajo Futuras**: Resume los resultados alcanzados, las lecciones aprendidas y propone posibles mejoras y extensiones para el proyecto.

En conclusión, este proyecto pretende no solo resolver un problema específico del análisis de trayectorias, sino también sentar las bases para futuras investigaciones y desarrollos en el ámbito del Big Data aplicado a datos geoespaciales.

2. Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

2.1. Objetivos generales

- Implementar un algoritmo de Big Data denominado TRA-CLUS para el análisis y agrupación de trayectorias geoespaciales.
- Facilitar la interpretación de los datos recogidos empleando sistemas de GPS mediante representaciones gráficas claras y precisas.
- Realizar comparativas de rendimiento y precisión entre diferentes algoritmos de clustering aplicados en combinación con el algoritmo TRA-CLUS.
- Diseñar y desarrollar una aplicación web interactiva para mostrar y analizar los resultados obtenidos por el algoritmo.

2.2. Objetivos técnicos

- Implementar el algoritmo TRA-CLUS optimizado para grandes volúmenes de datos utilizando herramientas eficientes y escalables.
- Diseñar y construir una aplicación web basada en *Dash* que soporte la visualización de datos y permita la interacción del usuario.

- Aplicar una arquitectura modular y bien documentada que facilite el mantenimiento y la extensibilidad del proyecto.
- Implementar herramientas de visualización como mapas interactivos para representar trayectorias y clusters.
- Aplicar metodologías ágiles como *Scrum* para organizar y gestionar las etapas del desarrollo del proyecto.
- Realizar pruebas unitarias, de integración y de interfaz para garantizar la calidad y fiabilidad del sistema.
- Utilizar *Git* como sistema de control de versiones distribuido y alojar el proyecto en la plataforma GitHub.
- Gestionar el desarrollo y el seguimiento del proyecto utilizando *GitHub Projects* para organizar tareas y monitorear el progreso.

2.3. Objetivos personales

- Realizar una contribución significativa al desarrollo experimental de software aplicado al análisis de Big Data.
- Reforzar y consolidar los conocimientos adquiridos durante la carrera en el ámbito del desarrollo de software y análisis de datos.
- Explorar y aplicar metodologías y herramientas actuales utilizadas en entornos laborales profesionales.
- Adentrarme en el campo del Big Data y el análisis geoespacial para comprender mejor sus desafíos y oportunidades.
- Profundizar en el desarrollo de aplicaciones web modernas que integren múltiples componentes tecnológicos.
- Mejorar habilidades en gestión de proyectos, trabajo colaborativo y resolución de problemas técnicos complejos.

3. Conceptos teóricos

En este apartado se presentan los conceptos teóricos fundamentales que permiten comprender el marco conceptual en el que se desarrolla este trabajo. Estos conceptos proporcionan el contexto necesario para el análisis y desarrollo del estudio realizado.

La discusión se centrará en los principios relacionados con el algoritmo TRACLUS, dado que este ha sido el enfoque principal del estudio y representa la mayor complejidad en su implementación y análisis.

3.1. Trayectorias GPS

Las trayectorias GPS se refieren a secuencias de puntos geoespaciales capturados mediante dispositivos de posicionamiento global (GPS). Cada punto de una trayectoria contiene información geográfica, como la latitud, longitud, y, en algunos casos, la altitud y el tiempo asociado. Estas trayectorias son fundamentales para analizar el movimiento de objetos o individuos a lo largo del tiempo y el espacio.

- **Formato típico**: Una trayectoria GPS suele representarse como una lista ordenada de coordenadas $[x, y]$, donde x corresponde a la longitud y y a la latitud. Un ejemplo típico es:

$$[[\text{longitud}_1, \text{latitud}_1], [\text{longitud}_2, \text{latitud}_2], \dots]$$

- **Origen de los datos**: Estas trayectorias se generan a partir de dispositivos móviles, vehículos, sensores de navegación y otros sistemas de rastreo que capturan posiciones geográficas en intervalos regulares de tiempo.

- **Aplicaciones**: Las trayectorias GPS son esenciales en áreas como la planificación de rutas, el análisis del tráfico, el monitoreo ambiental y la movilidad urbana. También sirven como base para algoritmos de agrupamiento y análisis de datos geoespaciales, como el algoritmo TRA-CLUS.

La importancia de las trayectorias GPS radica en su capacidad para modelar patrones de movimiento complejos, facilitando la identificación de tendencias, comportamientos y anomalías en contextos espaciales y temporales. Sin embargo, su análisis presenta desafíos debido a la densidad y complejidad de los datos.

3.2. Clustering

El **clustering** o agrupamiento es una técnica de aprendizaje no supervisado utilizada para organizar datos en grupos o çlusters"basados en características similares. Cada *cluster* está compuesto por elementos más similares entre sí que a elementos de otros *clusters*. Esta técnica es esencial en análisis exploratorio, permitiendo descubrir estructuras subyacentes en grandes volúmenes de datos y encontrar patrones, sin necesidad de tener etiquetas o categorías predefinidas.

En el contexto de análisis de datos, el clustering se aplica en múltiples áreas como la segmentación de clientes, detección de patrones de comportamiento, agrupación de imágenes y análisis de redes sociales. Entre los métodos de clustering más utilizados en la práctica se incluyen algoritmos basados en densidad y en conectividad, que proporcionan flexibilidad y adaptabilidad para manejar datos complejos y de alta dimensionalidad.

Clustering en el TRA-CLUS

En el algoritmo TRA-CLUS, el proceso de agrupamiento o *clustering* es una parte fundamental que se utiliza para identificar trayectorias similares basándose en los segmentos generados tras la partición de las trayectorias originales. Para este proyecto, se considera la posibilidad de sustituir el método de agrupamiento original de TRA-CLUS por algoritmos más avanzados o específicos que permitan optimizar los resultados.

La elección de un algoritmo de clustering para TRA-CLUS debe cumplir con las siguientes condiciones clave:

- **Métrica de Distancia Predefinida**: El algoritmo debe ser compatible con una métrica de similitud o distancia previamente calculada, es decir, debe aceptar matrices de distancias precomputadas. Esto es esencial, ya que la similitud entre segmentos en TRA-CLUS se mide previamente y se organiza en una matriz de distancias.

- **Adaptación a Densidades Variables**: Dado que las trayectorias suelen estar distribuidas en áreas con densidades variables, el algoritmo debe ser capaz de manejar estas diferencias para evitar sesgos en los resultados de agrupamiento.

- **Eficiencia Computacional**: Dado el volumen potencialmente grande de datos en estudios de trayectorias, el algoritmo debe ser computacionalmente eficiente para garantizar tiempos de procesamiento razonables.

Algoritmos de Clustering en scikit-learn

A continuación, se presenta una descripción de los algoritmos de clustering que cumplen las condiciones anteriores de la biblioteca `scikit-learn` de Python:

1. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN es un algoritmo basado en densidad que agrupa puntos que están en áreas de alta densidad y considera como ruido aquellos que se encuentran en áreas de baja densidad. Los clusters se forman alrededor de puntos densamente conectados y son identificados por dos parámetros: el radio de vecindad (ϵ) y el número mínimo de puntos necesarios ($minPts$) para formar un cluster. DBSCAN es especialmente útil para datos con formas irregulares y ruido, ya que ignora puntos aislados que no pertenecen a ninguna agrupación significativa.

2. OPTICS (Ordering Points To Identify the Clustering Structure)

OPTICS es una extensión de DBSCAN que aborda el problema de la sensibilidad a la elección de ϵ. En lugar de identificar clusters individuales directamente, OPTICS produce una ordenación de los puntos que muestra su estructura de densidad subyacente. Esto permite descubrir clusters a múl-

tiples escalas y niveles de densidad, haciendo posible una mayor flexibilidad en la agrupación de datos.

3. HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise)

HDBSCAN es una variante jerárquica de DBSCAN que forma clusters de densidad utilizando una estructura jerárquica en lugar de depender de un valor fijo de ϵ. A diferencia de DBSCAN y OPTICS, HDBSCAN construye una jerarquía de clusters que permite identificar agrupaciones en diferentes niveles de densidad sin requerir parámetros estrictos. Este algoritmo es particularmente útil cuando la densidad de los clusters varía significativamente.

4. Spectral Clustering

Spectral Clustering es un algoritmo de agrupación basado en teoría de grafos y técnicas de álgebra lineal. Utiliza los valores propios de una matriz de similitud de los datos para realizar la agrupación. Este enfoque es particularmente adecuado para datos que presentan estructuras de clusters no lineales o formas complejas. Spectral Clustering convierte el problema de agrupación en uno de corte de grafos, dividiendo el conjunto de datos en k clusters minimizando la similitud entre los clusters.

5. Agglomerative Clustering

El **Agglomerative Clustering** es una técnica jerárquica de clustering donde cada punto comienza como su propio cluster, y los clusters se fusionan iterativamente en función de una métrica de distancia (como la distancia euclidiana, de Manhattan, o de enlace promedio) hasta que se alcanza el número deseado de clusters o se completa la jerarquía. Este método es particularmente útil cuando se requiere una representación visual de los clusters en forma de dendrograma.

Comparación de los Algoritmos de Clustering

Los algoritmos de clustering mencionados se diferencian principalmente en su enfoque de agrupación (por densidad, jerárquico o basado en similitud), su sensibilidad a la elección de parámetros y su capacidad para manejar clusters de diferentes formas y densidades. A continuación, se presenta una tabla comparativa de los algoritmos:

Algoritmo	**Tipo de Clustering**	**Ventaja Principal**	**Limitación Principal**
DBSCAN	Densidad	Maneja ruido y clusters de formas arbitrarias	Sensible a la elección de ϵ y *minPts*
OPTICS	Densidad	Detecta clusters a diferentes niveles de densidad	Complejo de interpretar
HDBSCAN	Jerárquico basado en densidad	Sin parámetros estrictos	Computacionalmente costoso
Spectral Clustering	Basado en grafos	Captura estructuras complejas	Requiere especificar k
Agglomerative Clustering	Jerárquico	Ofrece dendrograma jerárquico	Alta complejidad para datos grandes

Conclusión

Los algoritmos basados en densidad, como DBSCAN, OPTICS y HDBSCAN, son especialmente útiles para datos con clusters de formas irregulares y en presencia de ruido, aunque requieren ajustes específicos de parámetros o presentan alta complejidad computacional. Por otro lado, métodos como Spectral Clustering y Agglomerative Clustering son más adecuados para datos que exhiben estructuras jerárquicas o formas no lineales, pero a costa de una mayor sensibilidad a la configuración inicial y un mayor costo computacional en datasets grandes.

En el contexto del algoritmo TRA-CLUS, donde la similitud entre segmentos se precomputa y los datos presentan densidades variables, los algoritmos basados en densidad, como HDBSCAN, ofrecen ventajas significativas por su flexibilidad y capacidad para manejar variaciones en los datos sin requerir parámetros estrictos. Sin embargo, la elección del algoritmo debe balancear la precisión de los resultados con la eficiencia computacional, considerando las características específicas de las trayectorias a analizar.

3.3. TRACLUS

TRACLUS [3] es un algoritmo de agrupación (*clustering*) especializado en datos de trayectorias, diseñado para identificar patrones comunes de movimiento en conjuntos de datos de trayectorias, como rutas de vehículos, movimientos de animales o trayectorias de fenómenos meteorológicos. A diferencia de los algoritmos tradicionales de *clustering* que agrupan pun-

tos individuales, TRACLUS se enfoca en el agrupamiento de trayectorias completas, descomponiendo cada trayectoria en segmentos y detectando patrones comunes en subtrayectorias específicas. Este enfoque es útil en estudios donde los objetos presentan secuencias de movimiento en el espacio y el tiempo, permitiendo identificar similitudes parciales dentro de grandes volúmenes de datos.

Principios y Funcionamiento de TRACLUS

El funcionamiento de TRACLUS se basa en dos etapas principales: la segmentación de trayectorias y la agrupación de segmentos de trayectorias. Ambas etapas están diseñadas para abordar la naturaleza secuencial y direccional de las trayectorias, empleando un enfoque basado en densidad que permite una identificación precisa de subtrayectorias similares.

1. **Segmentación de Trayectorias**: La primera etapa de TRACLUS es dividir cada trayectoria en segmentos de línea más cortos en función de cambios direccionales o puntos de inflexión. Estos puntos característicos dividen la trayectoria en subtrayectorias que pueden ser más fácilmente comparables. Este paso es crucial porque permite detectar patrones comunes en segmentos específicos, en lugar de requerir una coincidencia exacta en toda la trayectoria.

2. **Agrupación Basada en Densidad de Segmentos**: En lugar de agrupar puntos aislados, TRACLUS agrupa segmentos que se encuentran en regiones densas del espacio de trayectoria mediante una adaptación del algoritmo DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*). Esta agrupación basada en densidad identifica áreas de alta concentración de segmentos similares que constituyen patrones de movimiento comunes. Los clusters se forman en áreas de alta densidad de segmentos, separadas por regiones de baja densidad, permitiendo agrupar segmentos que comparten características similares, incluso si otras partes de la trayectoria son diferentes.

3. **Representación de Trayectorias**: TRACLUS utiliza los *clusters* de segmentos generados para representar patrones comunes de trayectorias. En lugar de visualizar cada segmento individual, los *clusters* se representan mediante trayectorias representativas que resumen las características de todos los segmentos pertenecientes a un mismo grupo. Estas trayectorias representativas se calculan promediando las posiciones y direcciones de los segmentos agrupados, proporcionando

una visualización simplificada y efectiva de los patrones de movimiento encontrados.

Métrica de Similitud en TRACLUS

TRACLUS emplea una métrica de similitud diseñada específicamente para medir la relación entre segmentos individuales, en lugar de comparar trayectorias completas. Esta métrica evalúa la similitud entre segmentos considerando tres aspectos principales:

- **Distancia perpendicular**: Mide la distancia más corta desde un punto de un segmento hasta la línea definida por el otro segmento. Esta medida capta la proximidad entre los segmentos desde una perspectiva geométrica.
- **Distancia paralela**: Evalúa la proyección de un segmento sobre el otro, midiendo cuánto de un segmento está alineado en la misma dirección que el otro. Esto permite identificar segmentos con orientaciones similares.
- **Diferencia en la longitud**: Compara las longitudes de los segmentos, lo que ayuda a identificar similitudes en términos de escala o tamaño.

La métrica de similitud en TRACLUS combina estas tres distancias en una medida agregada, lo que permite capturar tanto la proximidad espacial como la alineación y el tamaño relativo entre segmentos. Este enfoque es ideal para analizar conjuntos de datos con trayectorias complejas, ya que facilita la identificación de patrones subyacentes al dividir trayectorias en segmentos más pequeños. Sin embargo, este enfoque también conlleva consecuencias importantes en términos de complejidad computacional. Comparar las tres distancias para cada par de segmentos creados implica que cada una de estas mediciones tiene una complejidad de $O(n^2)$, donde n es el número total de segmentos. Esta complejidad cuadrática puede afectar significativamente el rendimiento cuando se trabaja con conjuntos de datos de gran tamaño, haciendo fundamental la optimización de las operaciones de comparación para mejorar la eficiencia general del algoritmo.

A diferencia de otras métricas como DTW (Dynamic Time Warping) o LCSS (Longest Common Subsequence), que se enfocan en medir la similitud entre series temporales completas, la métrica de TRACLUS proporciona

mayor flexibilidad al detectar subtrayectorias similares. Esto resulta especialmente útil para aplicaciones que requieren identificar patrones locales dentro de trayectorias extensas y densas.

Ventajas de TRACLUS en el Análisis de Trayectorias

El algoritmo TRACLUS ofrece varias ventajas que lo hacen adecuado para el análisis de datos de trayectoria:

- **Identificación de Subtrayectorias Similares**: TRACLUS no se limita a identificar patrones en trayectorias completas, sino que permite detectar similitudes en segmentos específicos. Esta capacidad es esencial en contextos donde solo algunas secciones de las trayectorias son comparables, mientras que otras presentan variaciones.

- **Adaptación a Escalas Variables**: Aunque TRACLUS es sensible a la densidad debido a su enfoque basado en clustering por densidad, su capacidad para manejar variaciones en la escala de las trayectorias lo hace útil en aplicaciones heterogéneas, como estudios de tráfico o movimientos de fauna en diferentes ecosistemas.

- **Selección Automática de Parámetros**: Mediante el uso de heurísticas para definir automáticamente los valores de parámetros clave (como el radio de vecindad ϵ y el número mínimo de puntos vecinos), TRACLUS reduce la necesidad de ajustes manuales. Sin embargo, la elección adecuada de estos parámetros sigue siendo crucial para garantizar resultados precisos en diferentes conjuntos de datos.

Aplicaciones de TRACLUS en la Investigación

TRACLUS se puede aplicar en múltiples campos de investigación, donde el análisis de patrones de movimiento es fundamental:

- **Biología y Ecología**: Para analizar trayectorias de animales, identificando patrones de comportamiento, rutas migratorias o territorios de caza.

- **Meteorología**: Para el estudio de trayectorias de fenómenos climáticos como huracanes, permitiendo identificar patrones comunes en ciertos eventos.

- **Gestión del Tráfico y Transporte**: En el análisis de rutas vehiculares, detectando patrones de congestión, flujo de tráfico y rutas populares.

4. Técnicas y herramientas

4.1. Introducción

En esta sección se presentan las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Se han considerado diferentes alternativas de metodologías y herramientas, y se ofrece un resumen de los aspectos más destacados de cada opción, junto con una justificación de las elecciones realizadas.

4.2. Python

Python es un lenguaje de programación versátil y fácil de aprender, ampliamente utilizado en el desarrollo de aplicaciones web, análisis de datos y machine learning. En este proyecto, se han utilizado varias bibliotecas de Python que han enriquecido el desarrollo y la funcionalidad de la aplicación:

Dash

Dash es un marco de trabajo desarrollado por Plotly que permite la creación de aplicaciones web analíticas e interactivas utilizando Python. Es especialmente popular en la comunidad de ciencia de datos y visualización debido a sus características y beneficios:

- **Interactividad:** Dash permite crear aplicaciones que responden a las interacciones del usuario, como clics, desplazamientos y entradas de datos. Esto es crucial para el análisis de datos en tiempo real y la visualización interactiva.

- **Integración con Plotly:** Las visualizaciones de Dash se basan en la biblioteca Plotly, que permite crear gráficos complejos y visualizaciones de alta calidad con facilidad. Esto enriquece la presentación de datos y facilita la comunicación de resultados.
- **Composición de componentes:** Dash permite combinar diferentes componentes (gráficos, tablas, controles de entrada) en una sola interfaz, lo que facilita la creación de aplicaciones integrales que ofrecen una experiencia de usuario fluida.
- **Despliegue sencillo:** Las aplicaciones construidas con Dash se pueden desplegar fácilmente en servidores web, lo que permite compartir los resultados del análisis con un público más amplio sin requerir instalación adicional por parte del usuario final.
- **Flexibilidad:** Al estar basado en Python, los desarrolladores pueden aprovechar la amplia gama de bibliotecas disponibles para manipular datos, realizar análisis y crear visualizaciones personalizadas, lo que proporciona gran flexibilidad en el desarrollo de aplicaciones.

Dash fue elegido para este proyecto debido a su idoneidad en la integración de visualizaciones interactivas con análisis de datos, permitiendo la creación de una interfaz eficiente y dinámica para los usuarios.

scikit-learn

Scikit-learn es una biblioteca robusta y eficiente para machine learning en Python. Proporciona herramientas de análisis predictivo, clustering y aprendizaje automático supervisado y no supervisado. Los algoritmos y parámetros utilizados en este proyecto incluyen:

1. **OPTICS (Ordering Points To Identify the Clustering Structure):** Este algoritmo identifica clusters de diferentes densidades en los datos. Los principales parámetros configurables son:
    a) **Metric:** Define la métrica utilizada para calcular las distancias entre puntos. Las opciones disponibles son:
        1) **euclidean:** Utiliza la distancia euclidiana tradicional.
        2) **l1:** Calcula la distancia Manhattan.
        3) **l2:** Similar a euclidean, pero con ajustes para ciertas aplicaciones.

4) **manhattan:** Igual que *l1*, mide la distancia entre dos puntos como la suma de sus diferencias absolutas.
5) **cosine:** Calcula la similitud basada en el coseno del ángulo entre vectores.
6) **cityblock:** Otra forma de describir la distancia Manhattan.

b) **Algorithm:** Selecciona el método para encontrar vecinos más cercanos:

1) **auto:** Elige automáticamente el método más eficiente según los datos.
2) **ball_tree:** Usa una estructura de árbol espacial para organizar los puntos.
3) **kd_tree:** Similar a *ball_tree*, pero optimizado para ciertas métricas.
4) **brute:** Calcula las distancias directamente, sin estructuras optimizadas.

c) **Max_eps:** Establece el radio máximo para considerar puntos vecinos.

d) **Min_samples:** Define el número mínimo de puntos necesarios para formar un cluster.

2. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):** Este algoritmo agrupa puntos densos mientras ignora ruido. Comparte parámetros con OPTICS:

a) **Metric:** Igual que en OPTICS.
b) **Algorithm:** Igual que en OPTICS.
c) **Eps:** Especifica el radio para considerar vecinos.
d) **Min_samples:** Igual que en OPTICS.

3. **HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise):** Este algoritmo extiende DBSCAN para trabajar mejor con datos de densidad variable:

a) **Metric:** Igual que en OPTICS.
b) **Algorithm:** Igual que en OPTICS.
c) **Min_samples:** Igual que en OPTICS.

4. **Agglomerative Clustering:** Este algoritmo agrupa puntos en una jerarquía utilizando diferentes criterios de vinculación:

   a) **Metric:** Igual que en OPTICS.
   b) **Linkage:** Determina cómo calcular la distancia entre clusters:
      1) **ward:** Minimiza la varianza dentro de los clusters.
      2) **complete:** Maximiza la distancia entre puntos más lejanos de clusters diferentes.
      3) **average:** Calcula la distancia media entre todos los puntos de dos clusters.
      4) **single:** Minimiza la distancia entre los puntos más cercanos de dos clusters.
   c) **n_clusters:** Especifica el número de clusters deseados en los resultados.

5. **Spectral Clustering:** Este algoritmo utiliza el espectro del grafo de afinidad para realizar clustering:
   a) **Affinity:** Define cómo construir el grafo de afinidad:
      1) **nearest_neighbors:** Usa vecinos más cercanos para definir las relaciones.
      2) **rbf:** Utiliza una función base radial para calcular similitudes.
      3) **precomputed:** Trabaja con una matriz de afinidad ya calculada.
      4) **precomputed_nearest_neighbors:** Usa vecinos más cercanos predefinidos.
   b) **Assign_labels:** Define el método para asignar etiquetas a clusters:
      1) **kmeans:** Usa el algoritmo K-Means para asignar etiquetas.
      2) **discretize:** Utiliza discretización para asignar etiquetas.
      3) **cluster_qr:** Asigna etiquetas usando descomposición QR.
   c) **n_clusters:** Especifica el número de clusters deseados en los resultados.

Otras librerías

Además de las bibliotecas principales, se emplearon diversas librerías complementarias que cubrieron necesidades específicas del proyecto. Estas se pueden agrupar en diferentes categorías según su funcionalidad:

Análisis y manipulación de datos

- **NumPy:** Una biblioteca fundamental para cálculos numéricos que proporciona soporte para matrices y funciones matemáticas. Es la base de muchas otras librerías de análisis y aprendizaje automático.
- **Pandas:** Ampliamente utilizado para trabajar con datos tabulares, se integra perfectamente con otras bibliotecas como NumPy y GeoPandas, facilitando tareas de limpieza, transformación y análisis de datos.
- **GeoPandas:** Extiende las capacidades de Pandas para trabajar con datos geoespaciales, permitiendo análisis y visualizaciones de datos basados en ubicación.
- **Shapely:** Diseñada para manipular y analizar geometrías en Python, es crucial para realizar operaciones como intersecciones y uniones de formas en análisis geoespacial.
- **pyproj:** Facilita las transformaciones de coordenadas y es esencial para interoperar entre distintos sistemas de referencia espacial.

Visualización de datos

- **Matplotlib:** Una biblioteca versátil para crear visualizaciones estáticas, animadas e interactivas, esencial para la comunicación de resultados.
- **Contextily:** Permite añadir mapas de fondo a visualizaciones geográficas, mejorando la contextualización e interpretación de datos espaciales.

Gestión de datos y archivos

- **JSON:** Proporciona herramientas para manipular datos en formato JSON, lo que facilita la interacción con APIs y configuraciones.
- **Zipfile:** Utilizada para gestionar archivos comprimidos ZIP, mejorando la transferencia y almacenamiento eficiente de datos.
- **io:** Ofrece funciones para manejar flujos de entrada y salida, útil para la manipulación de archivos y datos en memoria.

- **Base64:** Facilita la codificación y decodificación de datos en formato Base64, útil para transmitir archivos binarios como imágenes o documentos en formatos textuales.

Optimización y concurrencia

- **Time y threading:** Herramientas integradas en Python para gestionar temporización y tareas concurrentes, lo que mejora la eficiencia y la capacidad de respuesta de la aplicación.

- **ThreadPoolExecutor y Process (módulos concurrent.futures y multiprocessing):** Estas herramientas permiten ejecutar tareas intensivas de manera concurrente y en paralelo, mejorando el rendimiento general. Se utilizan para evitar bloqueos en el flujo principal y optimizar la ejecución de operaciones costosas en términos de computación.

4.3. CSS

Cascading Style Sheets (CSS) es un lenguaje utilizado para describir la presentación de documentos HTML y XML. En este proyecto, CSS se ha utilizado para mejorar la estética y la usabilidad de la interfaz de la aplicación desarrollada en Python con Dash. Las razones para su elección incluyen:

- **Separación de contenido y estilo:** CSS permite mantener el contenido HTML separado de su presentación, lo que facilita el mantenimiento del código y la implementación de cambios en el diseño sin afectar el contenido.

- **Responsividad:** Facilita el diseño responsivo, asegurando que la aplicación se vea bien en diferentes dispositivos y tamaños de pantalla. Esto es crucial para mejorar la accesibilidad y la experiencia del usuario.

- **Personalización:** Proporciona flexibilidad para personalizar la apariencia de la aplicación de manera sencilla, permitiendo la creación de un diseño atractivo y funcional que se alinee con los objetivos del proyecto.

- **Compatibilidad:** CSS es compatible con todos los navegadores modernos, lo que asegura que el diseño se mantenga consistente en diferentes plataformas.

Estas herramientas y bibliotecas han sido elegidas por su capacidad para facilitar el desarrollo, mejorar la eficiencia del trabajo y proporcionar funcionalidades que son fundamentales para el éxito del proyecto. Cada una de ellas contribuye a un enfoque integral que permite abordar las necesidades del análisis de datos y la creación de aplicaciones web interactivas.

4.4. Texmaker

Texmaker es un editor de texto multiplataforma para la creación de documentos en LaTeX. Es una herramienta esencial para la redacción académica y técnica, y ha sido fundamental en la redacción de este trabajo por las siguientes razones:

- **Interfaz amigable:** Proporciona un entorno intuitivo y fácil de usar que facilita la edición de documentos en LaTeX, incluso para aquellos que son nuevos en el sistema.
- **Compilación rápida:** Permite compilar documentos LaTeXrápidamente con un solo clic, lo que agiliza el proceso de revisión y mejora la productividad.
- **Herramientas integradas:** Incluye herramientas para la gestión de bibliografías, la inserción de gráficos y tablas, así como un visor PDF integrado que facilita la revisión del documento final.
- **Plantillas y ejemplos:** Ofrece diversas plantillas y ejemplos que ayudan a los usuarios a comenzar rápidamente con sus documentos, promoviendo buenas prácticas en la redacción científica.

4.5. GitHub

GitHub es una plataforma de desarrollo colaborativo basada en la web que utiliza el sistema de control de versiones Git. Es ampliamente utilizado en la comunidad de desarrollo de software por varias razones:

- **Control de versiones:** GitHub facilita el seguimiento de cambios en el código, lo que permite a los desarrolladores revertir a versiones anteriores si es necesario. Esto es esencial para la gestión de errores y la mejora continua del software.
- **Integración continua:** GitHub se puede integrar con diversas herramientas de automatización, como GitHub Actions, que facilitan la construcción, prueba y despliegue automático del código, mejorando la eficiencia del flujo de trabajo.
- **Documentación y seguimiento de issues:** Proporciona herramientas para documentar el código y gestionar tareas o problemas a través de un sistema de "issues", lo que facilita la organización y planificación del desarrollo.

4.6. Herramientas de control de tareas: Kanban y Scrum

Para llevar un control efectivo de las tareas y asegurar la evolución continua del proyecto, se han utilizado metodologías ágiles como **Kanban** y **Scrum**.

- **Kanban:** Se utilizó Kanban debido a su flexibilidad y enfoque visual. Kanban nos permitió gestionar el flujo de trabajo mediante tarjetas en un tablero visual, donde las tareas se movían de una columna a otra (de "Por hacer.ª .En progresoz "Hecho"). Esto facilitó una vista clara de las tareas pendientes, así como una priorización y reestructuración rápida de las mismas según los requerimientos que surgían.
- **Scrum:** Se implementó Scrum en ciclos de trabajo denominados *sprints*, con duraciones de dos semanas. Cada *sprint* comenzaba con una reunión en la que se revisaban en retrospectiva las tareas realizadas el último *sprint* y se planificaban los objetivos y metas para la siguiente reunión. Este ciclo permitió adaptar las prioridades en función de los avances realizados y los desafíos encontrados, proporcionando un proceso estructurado que se ajustó bien a las necesidades de fases más avanzadas del desarrollo.

El uso combinado de ambas metodologías facilitó tanto la flexibilidad como la estructura necesarias en diferentes fases del proyecto, contribuyendo a un flujo de trabajo eficiente y orientado a resultados.

4.7. Patrón Vista-Controlador

El patrón Vista-Controlador (MVC, por sus siglas en inglés) es un modelo arquitectónico ampliamente utilizado en el desarrollo de aplicaciones que separa la lógica de negocio, la lógica de presentación y la interacción del usuario en tres componentes distintos: Modelo, Vista y Controlador. En este proyecto, se adoptó una variante adaptada del patrón MVC para organizar y estructurar la aplicación desarrollada con Dash.

- **Modelo:** Representa la lógica de datos de la aplicación y maneja la interacción con el conjunto de datos, incluyendo el procesamiento de los mismos y su transformación para ser utilizados en la interfaz. Esto incluye las operaciones realizadas con bibliotecas como Pandas y GeoPandas.
- **Vista:** Es la responsable de la presentación visual de la aplicación. En este proyecto, las vistas fueron diseñadas utilizando Dash, con un enfoque en la interactividad y la visualización de datos, haciendo uso de componentes como gráficos y tablas.
- **Controlador:** Actúa como intermediario entre el modelo y la vista. En este proyecto, el controlador gestiona los callbacks de Dash, procesando las entradas del usuario y actualizando la vista en función de los cambios realizados en el modelo.

Esta separación de responsabilidades permitió un desarrollo modular, facilitando el mantenimiento y la escalabilidad del proyecto, al mismo tiempo que mejoró la claridad en la organización del código.

4.8. Testing: pytest y SonarCloud

El aseguramiento de la calidad del software es un aspecto crucial en el desarrollo de cualquier proyecto. En este trabajo, se implementaron pruebas automatizadas y herramientas de análisis de calidad del código para garantizar la robustez y el mantenimiento del sistema.

pytest

pytest es un marco de pruebas para Python que permite escribir casos de prueba de manera sencilla y eficiente. Las razones para su elección incluyen:

- **Simplicidad:** Permite escribir pruebas concisas utilizando una sintaxis clara y legible, lo que reduce el tiempo necesario para desarrollar y mantener los casos de prueba.

- **Cobertura de pruebas:** Facilita la ejecución de pruebas automatizadas para verificar la funcionalidad de diferentes módulos del proyecto, asegurando que los cambios en el código no introduzcan errores.

- **Extensibilidad:** pytest soporta la integración con complementos como *pytest-cov*, que proporciona informes detallados sobre la cobertura de código, ayudando a identificar áreas que requieren pruebas adicionales.

En este proyecto, se utilizaron pruebas unitarias y funcionales para validar la entrada y salida de datos, la interacción entre componentes y el correcto funcionamiento de los algoritmos de clustering y visualización.

SonarCloud

SonarCloud es una herramienta basada en la nube para analizar la calidad del código, identificar vulnerabilidades y mejorar la mantenibilidad del software. Su incorporación al proyecto permitió:

- **Análisis estático de código:** Identificar problemas como código duplicado, complejidad ciclomática y vulnerabilidades de seguridad antes de la ejecución.

- **Integración continua:** Integrar SonarCloud con GitHub para analizar automáticamente los cambios realizados en cada *commit*, proporcionando retroalimentación inmediata sobre la calidad del código.

- **Métricas de calidad:** Generar informes detallados sobre la calidad del código, ayudando a priorizar refactorizaciones y a mantener estándares de desarrollo altos.

La combinación de pytest y SonarCloud contribuyó a un enfoque integral para garantizar la calidad del proyecto, reduciendo errores y facilitando la detección temprana de problemas durante el desarrollo.

Sphinx

Sphinx es una herramienta de documentación automática que se utiliza principalmente para crear documentación técnica de proyectos de software. Originalmente fue diseñada para la documentación de Python, pero ahora se utiliza en una variedad de lenguajes y proyectos. Sphinx permite generar documentación en varios formatos, como HTML, LaTeX, ePub y PDF, a partir de archivos fuente escritos en reStructuredText (`.rst`) o Markdown.

Algunas de las características clave de Sphinx son:

- **Documentación basada en texto:** Utiliza un formato de texto plano fácil de escribir y leer (`reStructuredText`).
- **Soporte para código fuente:** Sphinx puede extraer automáticamente información de código fuente, como funciones y clases, a través de la directiva `autodoc`.
- **Generación automática de índices y tablas de contenido:** Sphinx crea automáticamente índices de contenido y enlaces internos, lo que mejora la navegación en la documentación.
- **Extensiones y personalización:** Sphinx es altamente extensible, permitiendo agregar funciones personalizadas mediante extensiones.
- **Soporte para múltiples formatos:** La documentación generada se puede exportar a diferentes formatos como HTML, PDF, LaTeX, y más.

En resumen, Sphinx es una herramienta poderosa y flexible para la creación de documentación de proyectos, que facilita la integración con el código fuente y la personalización del formato de salida.

5. Aspectos relevantes del desarrollo del proyecto

5.1. Formación y aprendizaje necesario

El desarrollo del proyecto requirió adquirir nuevos conocimientos y profundizar en diversas tecnologías y técnicas de análisis. A continuación, se describe la formación llevada a cabo en las herramientas y algoritmos más relevantes.

Estudio del algoritmo TRACLUS

El enfoque principal de este proyecto ha sido estudiar en profundidad el algoritmo **TRACLUS**, destinando una gran parte de las horas de investigación a comprender su funcionamiento y potencial para el análisis de trayectorias. Este algoritmo se basa en la segmentación y agrupación de trayectorias, permitiendo identificar sub-trayectorias comunes dentro de un conjunto de datos. Esto lo convierte en una herramienta valiosa para descubrir patrones significativos en datos de trayectorias. La investigación incluyó la revisión de artículos académicos y la exploración de cómo ajustar los parámetros de TRACLUS para maximizar su efectividad en el contexto específico del proyecto.

Adicionalmente, se evaluó el uso de una implementación de TRACLUS disponible en una biblioteca externa, explorando su viabilidad y características. Aunque esta versión fue útil para los primeros experimentos, se consideraron también posibles variantes y adaptaciones del algoritmo, con el fin de entender mejor el alcance y la adaptabilidad de TRACLUS.

Todos estos estudios fueron necesarios para cumplir con el objetivo del proyecto: implementar el algoritmo TRACLUS y compararlo con variantes del mismo, como las que se detallan a continuación.

Propuesta de variantes de TRACLUS

Durante el desarrollo del proyecto, se analizaron varias propuestas de variantes del algoritmo TRACLUS que han surgido en estudios recientes. Aunque finalmente no se utilizaron, debido al excesivo tiempo que llevaría desarrollar las nuevas variantes, la investigación detallada de estos derivados fue enriquecedora y ayudó a contrastar enfoques para una implementación eficiente. Las variantes de TRACLUS estudiadas fueron:

- **GTraclus:** Una variante diseñada para ejecutarse en unidades de procesamiento gráfico (GPU), optimizando la eficiencia del algoritmo al aprovechar la capacidad de procesamiento paralelo de las GPUs [6].
- **ST-TRACLUS:** Esta versión incorpora una dimensión temporal además de la espacial, permitiendo realizar agrupamientos espacio-temporales de trayectorias, lo cual mejora la calidad del análisis para datos donde el tiempo es una variable relevante [2].
- **ND-TRACLUS:** Una extensión de TRACLUS que permite realizar el clustering en espacios de n dimensiones, expandiendo su aplicabilidad a trayectorias de mayor dimensionalidad [4].
- **Neighborhood-Based Trajectory Clustering:** Una alternativa basada en densidad local de vecindad en lugar de densidad global. Este enfoque busca mantener la eficiencia de TRACLUS mientras reduce la necesidad de múltiples parámetros de entrada [**?**].
- **Adaptive Trajectory Clustering based on Grid and Density (ATCGD):** Un método que introduce una cuadrícula y criterios de densidad para el análisis de patrones móviles, buscando reducir la complejidad computacional y la carga de trabajo en la calibración de parámetros, especialmente en aplicaciones a gran escala como la de trayectorias de vehículos en sistemas de transporte inteligente [5].

Variación de las técnicas de clustering

Paralelamente a la investigación de las variantes de TRACLUS, se estudiaron técnicas adicionales de clustering, dado que el algoritmo requiere

la clusterización de segmentos para su evaluación final. Con el objetivo de evaluar la viabilidad y robustez de TRACLUS, se analizaron diversas técnicas de clustering, y tras descartar varias por inviabilidad, se optó finalmente por emplear los siguientes algoritmos ya incluidos en la librería de `scikit-learn`:

- **DBSCAN:** Algoritmo basado en densidad que identifica clusters de alta densidad separados por regiones de menor densidad, adecuado para datos espaciales y resistente al ruido.
- **OPTICS:** Similar a DBSCAN, pero permite una sensibilidad ajustable a la densidad, lo que lo hace más flexible para analizar datos con densidades variadas.
- **HDBSCAN:** Variante jerárquica de DBSCAN que ajusta automáticamente los parámetros de densidad, simplificando su aplicación en datos de densidad variable.
- **Spectral Clustering:** Algoritmo basado en el análisis de valores propios que es particularmente útil en datos no lineales o con clusters de formas complejas.
- **Agglomerative Clustering:** Método jerárquico ascendente que agrupa iterativamente elementos basándose en la proximidad, resultando útil en análisis donde la estructura jerárquica es relevante.

Cada técnica fue investigada en términos de sus características, sensibilidad a la densidad, capacidad de manejar ruido y aplicabilidad a datos espaciales. La exploración de estas técnicas permitió seleccionar aquellas que mejor se adaptaran a las necesidades específicas del proyecto y que complementaran el uso de TRACLUS en el análisis de trayectorias.

Página Web

El diseño de la página web comenzó con un prototipo básico que definía las funciones principales a implementar. La estructura inicial incluía:

- Una página para visualizar los datos sin procesar en forma de mapas de trayectorias y mapas de calor.

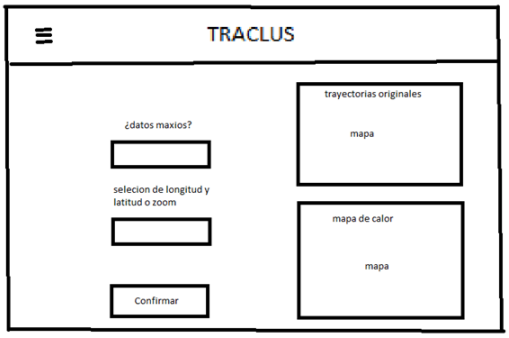

Figura 5.1: Código innecesario.

- Una segunda página dedicada a la comparativa de *clusters* mediante mapas interactivos.

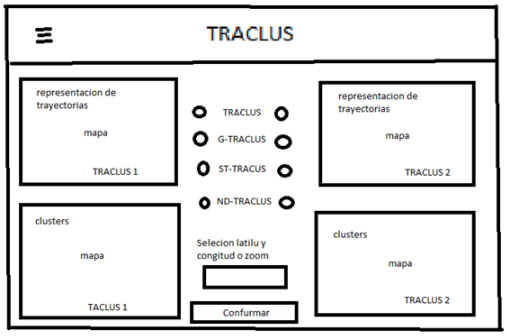

Figura 5.2: Código innecesario.

- Una tercera página para analizar la relación entre trayectorias y *clusters* generados por diferentes algoritmos.

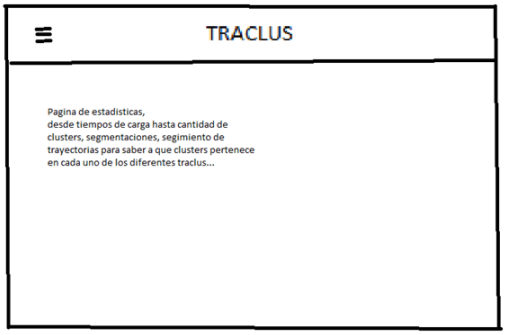

Figura 5.3: Código innecesario.

El desarrollo requirió integrar la biblioteca Dash con herramientas adicionales como `CSS Grid` para gestionar el diseño de los componentes. Para optimizar tiempos de carga, se decidió prerenderizar imágenes y tablas, limitando las opciones disponibles a configuraciones predefinidas.

Durante el proyecto, el diseño evolucionó para mejorar la experiencia del usuario. La página inicial se dividió en dos: una para la carga de datos y otra para la visualización de trayectorias. Además, se añadieron nuevas funcionalidades, como un botón para descargar datos generados y pantallas adicionales que permitían seleccionar algoritmos de *clustering* o reutilizar experimentos previos.

El menú desplegable inicial fue sustituido por una barra de navegación, lo que simplificó la interacción entre las diferentes páginas. También se eliminaron características que no aportaban valor significativo, como el zoom en los mapas, y se adoptó un enfoque basado en el modelo vista-controlador para estructurar el proyecto de manera más eficiente.

5.2. Optimización

El que podríamos considerar como el segundo punto más relevante, y sin duda uno de los aspectos críticos del proyecto, han sido los tiempos de ejecución. Debido a las grandes cantidades de datos a procesar, cada cálculo y visualización requerían una considerable cantidad de tiempo, por lo que la optimización de tiempos se convirtió en una prioridad clave para asegurar la viabilidad del proyecto.

Visualización y dibujado de Mapas

El primer desafió en el rendimiento fue la visualización de mapas, especialmente al intentar realizar representaciones detalladas y visualmente atractivas. Al analizar distintas bibliotecas de Python para dibujar y visualizar mapas, se probaron y compararon múltiples opciones combinadas entre si, como `pandas`, `geopandas`, `folium`, `folium.plugins`, `matplotlib.pyplot`, `matplotlib.colors`, `contextily`, `pyproj`, `seaborn`, `pydeck`, `shapely.geometry`, `sklearn.preprocessing` y `scipy.stats`.

Tras diversas pruebas de rendimiento y calidad visual, se optó por utilizar principalmente `contextily` y `matplotlib.pyplot` para la visualización de mapas, ya que estas bibliotecas ofrecieron la mejor relación entre rendimiento y calidad gráfica. Para la visualización de *heatmaps*, se incluyeron `numpy` como las opciones más eficientes para generar mapas de calor. Sin embargo, para lograr un equilibrio óptimo entre rendimiento y visualización, se sacrificaron ciertas funcionalidades como el zoom en mapas interactivos y detalles visuales avanzados.

Optimización del algoritmo

Uno de los principales desafíos del proyecto fue el tiempo de procesamiento, debido a la gran cantidad de datos que debían manejarse. El tamaño y la complejidad de los datos afectaron tanto la carga inicial como la ejecución de algoritmos y la visualización, requiriendo una optimización constante para lograr resultados en tiempos razonables.

Carga de Datos

El primer obstáculo en términos de tiempo fue la carga de datos en la aplicación web, ya que `Dash` tiene limitaciones al cargar grandes volúmenes de datos. El archivo inicial pesaba aproximadamente dos gigabytes, y ni siquiera herramientas como Excel podían manejar correctamente un CSV de este tamaño, lo que llevó a errores frecuentes al intentar reducir su tamaño sin comprometer la integridad de los datos.

Tras varios intentos de conversión y reducción, se logró un tamaño adecuado que permitió cargar los datos en la aplicación. Además, para estructurar y analizar los datos en trayectorias geográficas, fue necesario procesarlos con `GeoDataFrame` mediante la biblioteca `GeoPandas`. Gracias a esta biblioteca, el tiempo de carga y procesamiento fue moderado y permitió manipular los datos de forma eficiente para futuras visualizaciones y análisis.

Para la visualización, el tamaño del conjunto de datos impactó en la calidad gráfica y los tiempos de carga. Fue necesario optar por bibliotecas de visualización que equilibraran la calidad y el tiempo de renderizado, sacrificando algunas características visuales avanzadas en favor de un rendimiento adecuado. Finalmente, se optó por `contextily` y `matplotlib.pyplot` para la representación de mapas y `numpy` para los mapas de calor.

Distancias

El proceso más complejo y exigente en términos de tiempo fue la ejecución del algoritmo TRACLUS. Durante su ejecución, es necesario calcular las distancias perpendicular, paralela y angular entre todas las trayectorias, lo que genera una complejidad algorítmica exponencial $O(n^2)$. Esto se vuelve aún más costoso cuando se desean evaluar varias opciones de clustering en el mismo conjunto de datos.

Antes de las optimizaciones, el programa tardaba aproximadamente dos minutos y cuarenta y cinco segundos en cargar cien filas de datos, y una hora y cinco minutos en cargar quinientas filas, lo cual era insostenible dado que el archivo fuente de datos del proyecto contenía más de un millón de filas.

Para reducir estos tiempos, se realizaron múltiples pruebas y técnicas de optimización:

1. **Numpy para la matriz de distancia:** Inicialmente, se intentó cargar la matriz de distancia utilizando `numpy`, que es una biblioteca altamente optimizada para cálculos matemáticos. Sin embargo, los resultados generados no coincidían con los obtenidos mediante el cálculo original, lo que condujo a diferencias significativas en los resultados de clustering.

2. **Vectorización con Numpy:** En un segundo intento, se volvió a probar con `numpy`, aplicando un enfoque más completo de vectorización para las tres distancias. Nuevamente, aunque los tiempos de ejecución mejoraron, los resultados no fueron precisos, afectando la coherencia en los datos obtenidos.

3. **Threading con bucles:** Para el tercer intento, se reutilizaron las funciones originales de cálculo de distancias, pero paralelizando los tres bucles de cálculo de distancias mediante `threading`. Esta prueba mejoró los tiempos levemente, reduciendo la carga de cien filas a dos

minutos y veintidós segundos, manteniendo los resultados consistentes, aunque la mejora fue insuficiente para los objetivos del proyecto.

4. **ThreadPoolExecutor y chunks:** En la cuarta prueba, se dividió la matriz de distancia en "chunks" para ser procesados en paralelo con `ThreadPoolExecutor`. Sin embargo, esta técnica solo generó una mejora marginal, con una reducción de aproximadamente seis segundos en la carga de cien filas, lo cual, aunque significativo para cantidades de datos pequeñas, resultó ineficiente para volúmenes mayores.

5. **Paralelización manual y threading:** Finalmente, se probó una combinación de threading, sin usar `numpy` para los cambios en las funciones de distancia, calculando las distancias en tres hilos independientes. Este enfoque proporcionó el mejor resultado, logrando una reducción de tiempo a la mitad: un minuto y cuatro segundos para cien filas, y treinta y un minutos y cuarenta y nueve segundos para quinientas filas. Sin embargo, aunque esta reducción era significativa, los tiempos seguían siendo elevados para la totalidad de los datos del proyecto.

Este proceso de optimización requirió numerosos ajustes y pruebas incrementales, en muchos casos con cambios mínimos y variaciones en el tamaño de los datos de prueba, lo que demandó una gran cantidad de horas de prueba y error sin obtener los resultados esperados. No se tiene un cálculo exacto, pero este apartado del proyecto consumió no solo decenas, sino cientos de horas dedicadas exclusivamente a la ejecución y análisis de pruebas.

En conclusión, aunque Python es un lenguaje versátil, sus limitaciones en paralelización y threading efectivo complicaron la optimización de cálculos intensivos en comparación con otros lenguajes como Java, lo que limitó el rendimiento alcanzable en este proyecto.

Optimización de la página web

Tras múltiples ejecuciones de la página web para comparar los resultados de los diferentes algoritmos de clustering aplicados al TRACLUS, se identificó un problema recurrente: el tiempo de ejecución elevado.

En la aplicación, se podían ejecutar hasta cinco veces consecutivas el algoritmo TRACLUS, cada vez con modificaciones en el algoritmo de clustering. Esta ejecución lineal incrementaba significativamente los tiempos, ya que incluso con pocos datos, el algoritmo mostraba lentitud.

Estrategia inicial: Uso de hilos

Para abordar este problema, se reutilizó una estrategia previamente implementada: el uso de hilos. En el controlador `clustering.py`, que ya gestionaba correctamente el flujo de carga de datos, se dividió el código en funciones separadas, una por cada algoritmo de clustering. Estas funciones eran llamadas según las selecciones del usuario mediante la biblioteca `threading`.

Problemas con la generación de mapas y tablas

Posteriormente, se intentó incorporar la creación de mapas y tablas a estos hilos. Sin embargo, surgieron incompatibilidades con la biblioteca `matplotlib`, que no es compatible con librerías de hilos como `concurrent.futures` o `threading`. Por este motivo, se decidió separar estas tareas. Una vez finalizados todos los hilos, se invocaba una nueva función encargada de generar los mapas necesarios para visualizar los datos.

Limitaciones de Python

Aunque esta optimización redujo el tiempo de ejecución, los resultados no alcanzaron las expectativas iniciales. Se identificó que la principal limitación era Python. Su configuración interna impide utilizar múltiples núcleos del procesador de manera eficiente, lo que restringía la velocidad máxima en dispositivos locales. Sin embargo, estas limitaciones no afectaban significativamente la aplicación en entornos remotos, donde el alto rendimiento no era una prioridad debido a los costes asociados.

Exploración de alternativas de paralelización

Dado el potencial escalamiento de la aplicación, se exploraron otras estrategias de paralelización: - `asyncio`: Diseñada para tareas intensivas en I/O, pero no adecuada para este caso. - `Batch Processing`: Orientada a flujos simples, tampoco era aplicable. - `Dask` y `ProcessPoolExecutor`: Ambas opciones cumplían con los requisitos del proyecto.

Se optó por `ProcessPoolExecutor` debido a su menor complejidad y capacidad para ejecutar funciones en diferentes núcleos del procesador.

Optimización del flujo de datos

Además del uso de `multiprocessing`, se unificó el flujo de representación de los datos en las funciones de multiproceso, lo que resolvió incompati-

bilidades previas con `matplotlib`. Esto incrementó considerablemente el rendimiento, ya que anteriormente estas tareas se ejecutaban en serie.

Resultados obtenidos

Tras implementar estas mejoras, se lograron resultados significativos: - **Operaciones singulares**: La ejecución de un solo TRACLUS con 100 filas de datos pasó de 312 segundos a 288 segundos. - **Ejecuciones simultáneas**: La ejecución de cinco TRACLUS con sus respectivos algoritmos de clustering se redujo drásticamente, de 1800 segundos a 450 segundos.

Esta optimización no solo mejoró el rendimiento, sino que también eliminó errores previamente asociados al uso de hilos.

5.3. Testing

Para que un código sea verdaderamente funcional, seguro y mantenible, debe someterse a pruebas exhaustivas. Durante el desarrollo del proyecto, se implementaron varios tipos de pruebas para evaluar el rendimiento y la calidad del código, así como para ampliar el conocimiento en el campo del testing.

Análisis de código estático

El análisis de código estático es una técnica utilizada para identificar posibles errores en el código fuente sin necesidad de ejecutarlo. Este método es particularmente útil para evitar fallos humanos como bucles infinitos, errores de formato o problemas de nomenclatura.

Se evaluaron varias herramientas de análisis estático teniendo en cuenta las siguientes limitaciones:

- Debían ser compatibles con Python y CSS, los lenguajes utilizados en el proyecto.
- No debían implicar costos adicionales.
- Era deseable la integración directa con GitHub para facilitar el flujo de trabajo.

Con estas características, se seleccionaron dos herramientas principales: **SonarQube** y **Code Climate**. De estas, **SonarQube** fue la más utilizada,

ya que ofrecía soporte para analizar IPython Notebooks, un formato clave en los experimentos realizados durante el proyecto. Aunque los notebooks no formaban parte directa de la aplicación web, su análisis fue crucial para garantizar la calidad de los experimentos previos.

Lo ideal en este tipo de análisis es aplicarlo de manera continua durante las diversas fases del proyecto, ya que esto reduce significativamente la carga de trabajo y mejora la calidad del código a medida que se desarrolla. Sin embargo, en este proyecto, el análisis estático se realizó al final, lo que no fue óptimo pero permitió identificar una serie de problemas, entre ellos:

- **Mejoras en nomenclatura:** Se ajustaron nombres de variables para facilitar su comprensión y evitar confusiones con otros datos.

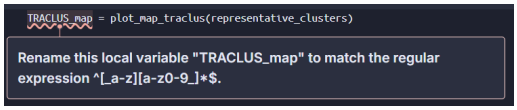

Figura 5.4: Error nomenclatura.

- **Variables no utilizadas:** Se detectaron y eliminaron variables que ya no eran relevantes, ya fuera por errores en su definición o por haber quedado obsoletas durante el desarrollo.

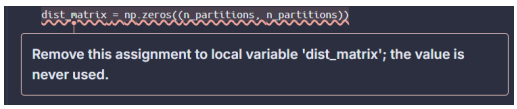

Figura 5.5: Código innecesario.

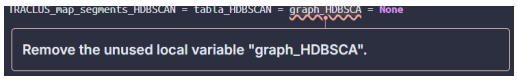

Figura 5.6: Variable sin usar necesaria.

- **Importaciones innecesarias:** Se eliminaron módulos y librerías no utilizadas, lo que ayudó a reducir la carga del programa y mejorar su legibilidad.
- **Complejidad en funciones:** Se identificaron funciones cuya complejidad excedía los límites recomendados. Aunque algunas de estas

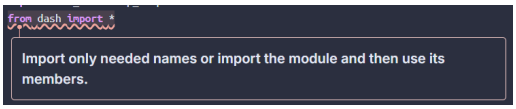

Figura 5.7: Importación excesiva.

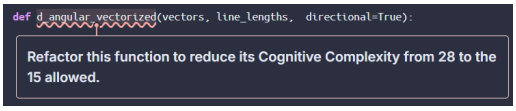

Figura 5.8: Complejidad grande.

funciones no pudieron simplificarse debido a las características del algoritmo, el análisis ayudó a priorizar futuras mejoras.

Aunque este tipo de análisis no corrige directamente errores en la funcionalidad del algoritmo, resulta muy útil para evitar problemas menores, mejorar la mantenibilidad del código y garantizar un nivel básico de seguridad.

Pruebas unitarias y su implementación

Las pruebas unitarias son fundamentales para garantizar la funcionalidad de cada componente del código y asegurar que los resultados generados sean consistentes y correctos. Durante el desarrollo de la aplicación, se planteó inicialmente realizar pruebas unitarias utilizando `pytest` que evaluaran el comportamiento de cada botón y función de la aplicación mientras esta se ejecutaba. El objetivo era cubrir todo el flujo de la web en tiempo real, desde la interacción del usuario con los botones hasta la ejecución de los `callbacks` de Dash.

Sin embargo, esta aproximación presentó múltiples problemas. Dash, como framework basado en componentes interactivos, no es directamente compatible con las herramientas de testing tradicionales. Intentar realizar pruebas unitarias mientras la aplicación se encontraba en ejecución resultó inviable, ya que las pruebas no se ejecutaban correctamente, y la interacción con los componentes de la interfaz gráfica no podía ser simulada adecuadamente. Como resultado, las pruebas terminaban siendo llamadas directas a las funciones asociadas a los botones, sin representar el comportamiento real del flujo de la aplicación ni garantizar que los `callbacks` funcionaran en contexto.

Ante esta limitación, se decidió cambiar el enfoque hacia pruebas más efectivas y relevantes. La nueva estrategia consistió en lo siguiente:

- **Pruebas de funciones críticas:** Se probaron de manera individual las funciones más importantes de los modelos, como la implementación del algoritmo TRACLUS y su integración con diferentes algoritmos de clustering. Estas pruebas se realizaron utilizando conjuntos de datos generados de manera aleatoria para garantizar que el comportamiento del código fuese robusto ante diferentes escenarios.
- **Representación de mapas:** Se verificó que las funciones encargadas de generar mapas, tanto de segmentos como de clústeres, produjeran respuestas correctas.

6. Trabajos relacionados

6.1. Introducción

En el desarrollo de este proyecto, fue crucial estudiar tanto los fundamentos teóricos como los trabajos previos relacionados con el algoritmo TRA-CLUS, ya que estos sirvieron como base para implementar y adaptar la solución propuesta. Este capítulo describe dos elementos clave: el artículo original que introduce TRA-CLUS y la biblioteca de código en GitHub que proporcionó una referencia práctica para la implementación del algoritmo.

6.2. Estudio del algoritmo TRA-CLUS

El algoritmo TRA-CLUS, introducido en el artículo *Trajectory Clustering: A Partition-and-Group Framework* [3], establece un marco novedoso para la agrupación de trayectorias basado en dos fases principales: segmentación y agrupamiento. Este enfoque busca identificar patrones significativos en conjuntos de datos espaciales y temporales, como los recorridos de taxis, rutas de navegación o trayectorias de animales.

Entre los conceptos fundamentales del algoritmo destacan:

- **Segmentación**: Las trayectorias se dividen en segmentos lineales, utilizando un enfoque de optimización que minimiza el error de representación.
- **Agrupamiento**: Los segmentos resultantes se agrupan utilizando un algoritmo de clustering basado en densidad, como DBSCAN, para identificar patrones comunes en las trayectorias.

- **Representación**: Cada grupo de segmentos se representa mediante una trayectoria representativa", que captura la esencia del grupo y permite una interpretación más clara de los datos.

Este marco de partición y agrupamiento demostró ser efectivo para manejar grandes volúmenes de datos geoespaciales, ofreciendo una solución escalable y precisa para el análisis de trayectorias. Este trabajo sirvió como base teórica para este proyecto, orientando el desarrollo y la implementación del algoritmo.

6.3. Biblioteca TRA-CLUS

Para complementar el estudio teórico del algoritmo, se utilizó como referencia práctica la biblioteca de código *TRA-CLUS* disponible en GitHub [1]. Esta implementación fue desarrollada por Adriel Amoguis, investigador asociado al Dr. Andrew L. Tan Data Science Institute (ALTDSI) [**?**], una institución reconocida por su enfoque en proyectos avanzados de ciencia de datos y aprendizaje automático. La experiencia del autor en el desarrollo de herramientas de análisis de datos y su afiliación con una institución de prestigio hacen de esta biblioteca una fuente confiable y valiosa. Además, su código presenta una estructura modular y clara, lo que facilita su reutilización y adaptación para proyectos personalizados, cumpliendo con los estándares de calidad esperados en implementaciones de algoritmos complejos como TRA-CLUS.

Características principales de la biblioteca

La biblioteca *TRA-CLUS* destaca por:

- Una implementación directa de las dos fases principales del algoritmo: segmentación y agrupamiento.
- Código bien documentado que facilita su comprensión y modificación.
- Ejemplos prácticos que demuestran su aplicación en datasets sencillos, lo que reduce la curva de aprendizaje para los usuarios.

Adaptaciones y mejoras realizadas en este proyecto

Aunque la biblioteca original ofrecía una base sólida, fue necesario realizar una serie de adaptaciones para cumplir con los requisitos específicos de este proyecto:

- **Optimización**: Se mejoraron ciertos aspectos del código para garantizar su rendimiento en conjuntos de datos más grandes, como los datos de trayectorias de taxis utilizados en este trabajo.
- **Integración con visualización**: Se desarrollaron herramientas adicionales para vincular los resultados del algoritmo con representaciones gráficas interactivas, facilitando la interpretación de los datos.
- **Extensibilidad**: Se modularizó aún más el código para permitir su integración en la aplicación web desarrollada.
- **Comparativas**: Se añadieron funciones para realizar comparaciones con otros algoritmos de clustering, como DBSCAN, OPTICS y HDBSCAN.

6.4. Impacto en el proyecto

El uso combinado del artículo original de TRA-CLUS y la biblioteca de código permitió acelerar el desarrollo y centrar los esfuerzos en aspectos diferenciadores del proyecto, como la visualización y el análisis comparativo. Además, este enfoque demostró cómo los trabajos previos pueden ser un recurso invaluable en la investigación y el desarrollo, ofreciendo tanto una base teórica sólida como herramientas prácticas para la implementación.

7. Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

El desarrollo de este proyecto ha permitido abordar diversas problemáticas relacionadas con el análisis y la visualización de datos geoespaciales mediante técnicas de clustering. A continuación, se resumen las principales conclusiones obtenidas durante su realización:

- Se logró implementar con éxito el algoritmo TRA-CLUS, permitiendo segmentar y agrupar trayectorias geográficas de manera eficiente, y generando resultados útiles para la interpretación de grandes volúmenes de datos.

- La aplicación web desarrollada ha demostrado ser una herramienta valiosa para visualizar, comparar y analizar los resultados de diferentes algoritmos de clustering. La posibilidad de interactuar con gráficos, mapas y tablas ha facilitado la interpretación de los datos.

- Durante las pruebas comparativas, se evidenció que cada algoritmo de clustering tiene ventajas y limitaciones dependiendo de las características de los datos y los parámetros configurados. Esto refuerza la importancia de disponer de herramientas flexibles que permitan explorar múltiples opciones.

- La adopción de la arquitectura modelo-vista-controlador (MVC) mejoró significativamente la organización y mantenibilidad del código,

facilitando la incorporación de nuevas funcionalidades y la resolución de problemas técnicos.

- El despliegue de la aplicación en un entorno de producción a través de la plataforma Render permitió validar la funcionalidad del sistema en condiciones reales, garantizando su accesibilidad a otros usuarios.

Técnicamente, el proyecto ha permitido afianzar conocimientos en áreas como el análisis de trayectorias, la programación en Python, el uso de bibliotecas de visualización y frameworks web, además de explorar técnicas de optimización y manejo de datos masivos.

7.2. Líneas de Trabajo Futuras

A pesar de los logros alcanzados, existen múltiples aspectos que podrían mejorarse o extenderse en futuros trabajos relacionados con este proyecto. Algunas de las líneas de trabajo futuras incluyen:

- **Optimización del rendimiento**: Implementar técnicas de optimización tanto en el procesamiento de datos como en la generación de gráficos interactivos para reducir los tiempos de carga y procesamiento en datasets más grandes.
- **Ampliación de funcionalidades**: Incluir soporte para nuevos algoritmos de clustering, como Birch o K-Means, y permitir configuraciones más avanzadas de los parámetros por parte del usuario.
- **Análisis en tiempo real**: Adaptar la aplicación para procesar datos en tiempo real, lo que sería especialmente útil en aplicaciones relacionadas con la gestión del tráfico o la monitorización de flotas de vehículos.
- **Mejoras en la interfaz de usuario**: Refinar la experiencia del usuario mediante el uso de bibliotecas avanzadas de diseño y optimización de la navegación, además de añadir herramientas de análisis más intuitivas.
- **Integración con otras fuentes de datos**: Permitir la carga y análisis de datos provenientes de diversas fuentes, como APIs de mapas en tiempo real, sensores o bases de datos externas.
- **Validación con expertos**: Realizar evaluaciones cualitativas con expertos en análisis geoespacial para identificar posibles mejoras en los resultados generados y la interfaz de la aplicación.

- **Publicación científica**: Documentar los resultados obtenidos y presentarlos en conferencias o revistas científicas relacionadas con el análisis de datos y Big Data.

En conclusión, este proyecto ha sentado las bases para el desarrollo de sistemas de análisis de trayectorias avanzados, demostrando la viabilidad y utilidad de combinar algoritmos de clustering con herramientas de visualización. Los resultados obtenidos abren un abanico de posibilidades para futuras investigaciones y desarrollos en el ámbito del Big Data aplicado a datos geoespaciales.

Bibliografía

[1] Adriel Amoguis. Tra-clus library, 2023. Último acceso: noviembre 2024.

[2] Hao Chen, Jing Yu, Yanhong Li, and Xiaofeng Yang. A trajectory clustering algorithm for spatio-temporal analysis. *Journal of Systems Engineering and Electronics*, 33(3):653–662, 2022. Último acceso: noviembre 2024.

[3] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: A partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 593–604. ACM, 2007.

[4] Mirco Nanni and Federico Ronzoni. Nd-traclus: a density-based trajectory clustering algorithm. *ISPRS International Journal of Geo-Information*, 7(1):25, 2018. Último acceso: noviembre 2024.

[5] Hongzhi Wang, Weizhi Song, Jinjun Cao, and Haitao Wang. Adaptive trajectory clustering based on grid and density. *Sensors*, 17(9):2013, 2017. Último acceso: noviembre 2024.

[6] Mostafa Yousefpour, Ali Ghaffari, and Ali Jabbari. Gtraclus: trajectory clustering approach for highly sparse spatio-temporal data. *Distributed and Parallel Databases*, 41:533–556, 2023. Último acceso: noviembre 2024.