

Prevención y predicción de accidentes de tráfico. Análisis mediante técnicas de machine learning y datos geoespaciales

31 Octubre 2024



Universidad
Internacional
de Valencia

Titulación:

Máster VIU en Big Data y
Ciencia de Datos

Curso académico

2023 – 2024

Alumno/a:

Nieto Royuela José María

D.N.I: 06587961W

Director/a de TFM:

Edith Cecilia Macedo Muiz

Convocatoria:

SEGUNDA
CONVOCATORIA

De:

 Planeta Formación y Universidades

Índice

Resumen.....	6
1. Introducción.....	7
1.1. Motivación.....	7
1.2. Planteamiento del problema.....	7
2. Objetivos.....	11
2.1. Objetivo general.....	11
2.2. Planteamiento.....	11
2.2.1. Planteamiento de los datos.....	11
2.2.2. Planteamiento del modelo.....	12
2.3. Objetivos específicos.....	12
3. Marco teórico y estado del Arte.....	13
3.1. Marco teórico.....	13
3.1.1. Redes neuronales.....	13
3.1.2. Capas LSTM.....	13
3.1.3. Redes de grafos.....	14
3.2. Estado del arte.....	14
4. Desarrollo del proyecto y resultados.....	23
4.1. Enunciado formal del proyecto.....	23
4.2. Metodología.....	23
Librerías utilizadas.....	23
4.2.1. Elección de datos geoespaciales relevantes.....	24
4.2.1.1. Datos de accidentes.....	24
4.2.1.2. Datos de tráfico.....	24
4.2.1.3. Datos meteorológicos.....	25
4.2.1.4. Datos laborales y sociales.....	25
4.2.1.5. Fuentes de datos.....	26
Portal de datos abiertos del Ayto. Madrid.....	26
API web OpenStreetMap.....	26
4.2.2. Estudio de modelos.....	26
4.2.3. ETL de los datos.....	26
4.2.3.1. Datos de Tráfico.....	26
4.2.3.2. Datos de accidentes.....	27
4.2.3.3. Datos meteorológicos.....	27
4.2.3.4. Datos Calendario.....	28
4.2.4. Desarrollo del grafo.....	28
4.2.5. Adaptación de datos para el modelo escogido.....	29
4.2.6. Justificación del modelo.....	30
4.3. Desarrollo del proyecto.....	32
4.3.1. Desarrollo del proyecto.....	35
4.3.1.1. Extracción, Transformación y Carga de los datos.....	35



4.3.1.2. Desarrollo del dataset.....	39
4.3.1.3. Desarrollo del modelo.....	47
4.3.2. Evaluación y resultados.....	52
4.3.3. Limitaciones, posibles mejoras y consideraciones.....	54
5. Referencias.....	56
Apéndice I : Código Dataset de Grafo Espacio temporal.....	61
Estructura de Ficheros.....	61
Carpeta Ubicaciones.....	62
Carpeta Sensores.....	62
main.py.....	63
funcs.py.....	63
Utils.py.....	63
CONSTANTS.py.....	63
Estructura del código principal main.py.....	64
Apéndice II : Código modelo predicción con Grafos.....	66
Anexos I : Documentación Fuentes de datos.....	68



Índice de ilustraciones

Ilustración 1. Arquitectura Jupyter Cliente - Servidor. Fuente:
<https://www.paradigmadigital.com/dev/jupyter-data-science-aplicada/> 4

!!!INFORMATIVO!!! (quitar)

**Actualizar índice de tablas (Referencias / Insertar tabla de ilustraciones)*

Índice de tablas

Tabla 1. Operaciones matemáticas utilizadas en el estudio realizado. Elaboración propia. 3

!!!INFORMATIVO!!! (quitar)

**Actualizar índice de tablas (Referencias / Insertar tabla de ilustraciones)*

Resumen

Los accidentes de tráfico en entornos urbanos plantea un desafío significativo para la seguridad vial y la gestión eficiente del tráfico. La complejidad de la red vial y las diversas condiciones ambientales complican la predicción precisa de estos eventos. En la era de la información y a pesar del avance de las herramientas inteligentes de predicción de tráfico y rutas de tránsito, existen pocos estudios relativos a la predicción de los accidentes con el uso de modelos de aprendizaje profundo.

En el presente documento, particularizando el caso para la ciudad de Madrid, se pretende crear un modelo en el que la variable de salida sean los accidentes de tráfico para un área dada, esto utilizando herramientas de Deep learning, Particularmente las redes neuronales de grafos temporales o T-GCN, que se han demostrado efectivas para manejar y analizar datos geoespaciales complejos, permitiendo captar relaciones espaciales y temporales cruciales para la predicción de accidentes. como pueden ser las redes neuronales de grafos (Nvidia Blog, 2022).

Para ello se han procesado fuentes de datos de diversa índole con el objetivo de ampliar las variables que pueden ser relevantes para este proceso integrándose en posición de la red viaria de la ciudad de Madrid.

Conclusiones.....

Palabras clave: Predicción de accidentes, Redes neuronales de grafos, Datos geoespaciales, Deep learning, Tráfico, Madrid.

1. Introducción

1.1. Motivación

La seguridad vial en entornos urbanos es un desafío creciente debido al aumento continuo de vehículos y la complejidad de las redes viales (Comunidad de Madrid 2022). Los trabajos existentes en el campo del análisis y predicción de datos de tráfico se han centrado predominantemente en la predicción del flujo vehicular y la congestión, dejando en segundo plano la predicción específica de accidentes de tráfico.

Tener un mejor conocimiento de los eventos de tráfico es crucial para reducir las tasas de mortalidad y lesiones, así como para mejorar la planificación urbana. Planificación en la cual está inmersa Madrid con la renovación de varios sectores de su red urbana e interurbana.

También cobra una especial importancia a la hora de la optimización de los recursos de emergencia y médicos necesarios en cada punto de la ciudad, sabiendo dónde se pueden ocasionar las mayores incidencias y su gravedad se pueden proveer mejor las dotaciones sanitarias y de seguridad colindantes en cada momento. En el caso de Madrid, también puede ser especialmente relevante ya que es una gran ciudad en crecimiento continuo y sus redes sanitarias cada vez se ven más afectadas y se deben optimizar al máximo los recursos para minimizar daños o incluso pérdidas humanas ya que los accidentes de tráfico suponen 4 de cada 10 de los ingresos graves (Comunidad de Madrid 2023).

1.2. Planteamiento del problema

La predicción de accidentes, a diferencia de la predicción de tráfico, aún estando íntimamente ligados, requiere una comprensión más profunda de las interacciones entre diversos factores, como las condiciones meteorológicas, el comportamiento de los conductores y la configuración de la infraestructura vial.

Por otro lado, las técnicas de deep learning, y en particular las redes neuronales, han demostrado ser herramientas potentes y versátiles para abordar problemas complejos con grandes volúmenes de datos mejorando a las herramientas de análisis clásicas como pudiera ser análisis ARIMA/SARIMA, (Sirisha et al., 2022), que también se observa una herramienta capaz (Deretić et al., 2022) aunque con sus limitaciones. Estas redes son capaces de extraer características relevantes y representaciones abstractas de los datos sin la necesidad de un extenso preprocesamiento manual. Esto es particularmente útil en el contexto de la predicción de accidentes, donde los factores influyentes pueden ser numerosos y heterogéneos.

Para diferentes tipos de predicciones son extensamente usadas las redes neuronales convolucionales, siendo varios los campos y variantes de interés dentro de los accidentes de tráfico (Pradhan & Sameen, 2019)

No obstante, las redes neuronales convolucionales, que son las que mayor incidencia han tenido recientemente con el desarrollo de reconocimiento y clasificación y segmentación de imagen, se estructuran y dependen de una estructura de datos regular, en forma de cuadrilla n-dimensional, normalmente 2 o 3 dimensional en el caso de imágenes u objetos 3D. No es este el caso de las redes viales de tráfico que segmentan y unen el espacio anisométricamente, es decir de manera irregular, debido a la propia condición de las vías de no repartirse de igual manera por el territorio y el espacio.

En este contexto cobra sentido la introducción del concepto de grafo para interconexionar la información según una información no regular de una manera mucho más relevante que unas coordenadas en un espacio euclidiano regular, ya que en una red de carreteras los accidentes y tráfico vial no está homogéneamente repartido por el espacio, sino que sólo ocurre en áreas donde hay carreteras una parte minoritaria en todo el área del mapa, de manera que la información fuera de las vías no aporta información es irrelevante. (Yuan, 2023)

Los grafos son puntos o nodos de datos interconectados de una manera concreta de manera que forman una estructura interdependiente. Esta estructura sería una generalización de la estructura regular de la que hemos hablado. Simplemente en la estructura regular todos los nodos tienen la misma relación con sus vecinos. (Scarcelli, 2009)

Por esta razón, la similitud y adecuación de la estructura de datos no regulares en la rama de información vial, las redes neuronales de grafos son una herramienta muy adecuada para el estudio que atañe a este estudio. En la imagen se observa representada una red de carreteras, los accidentes se presentarán en las líneas que representa la red de carreteras y el resto de coordenadas espaciales y píxeles no aportan al modelo, con lo cual en un grafo se está representando la información de manera mucho mejor comprimida y representada que en una imagen normal. El modelo de red de carreteras, es precisamente el de un grafo, donde las intersecciones son los nodos de la red y los tramos de carretera son las llamadas aristas.



Estas son algunas de las razones que posicionan a las GCN sobre las DNN para este campo de trabajo concreto, al representar a nivel de estructura de datos, de una manera más fiel el problema.

Modelado Natural: Las GNNs están diseñadas para manejar datos estructurados como grafos, lo que las hace ideales para modelar redes viales. Esto se debe a que pueden capturar las interacciones complejas y la topología variable de las redes viales. Las CNNs además, requieren que los datos se estructuren en una cuadrícula regular, lo que no se ajusta bien a la topología variable y compleja de una red viaria adaptándose a cambios y modificaciones..

Relaciones Espaciales Complejas: Las GNNs, gracias a su mejor comprensión del entorno, pueden capturar las relaciones espaciales complejas entre los nodos de un grafo, lo cual es crucial para entender cómo las diferentes partes de una red viaria interactúan entre sí. Esta capacidad supera las limitaciones de las CNNs, que no pueden manejar eficientemente conexiones de largo alcance y dependencias no locales

Propagación de Información: Las GNNs permiten que cada nodo en la red viaria "conozca" información de sus vecinos y más allá mediante la propagación de información a través de las aristas del grafo.

Representación Aprendida: Son capaces de aprender representaciones de los nodos del grafo capturando las características y relaciones más relevantes, mejorando la capacidad del modelo para hacer predicciones precisas sobre la red viaria

En definitiva, han demostrado ser más eficientes en el modelado de datos de tráfico, proporcionando predicciones más precisas al capturar mejor la estructura y dinámica de la red de carreteras. Esto contrasta con las CNNs, que a menudo requieren una partición de los datos en unidades espaciales regulares, lo que puede llevar a una

pérdida de información espacial importante.(K. Chen et al., 2020). Debido a que la bibliografía sobre el tema se centra principalmente en la predicción del tráfico y no en la estricta predicción de accidentes, al estar íntimamente ligadas y se basan espacialmente en los mismos principios se establece que los resultados podrían ser aptos para ser extrapolados a nuestro campo de los accidentes viales. Por tanto, se presentan evidencias que respaldan el desarrollo del presente documento a través de estos modelos.

Tras el despegue del aprendizaje profundo la ciencia y análisis de datos en los años recientes, son múltiples los estudios, publicaciones y trabajos realizados en diferentes ámbitos y enfoques relativos a los accidentes y al tráfico. En este documento se repasan estas publicaciones y en base a ellas se otorgará una justificación al análisis realizado.

Este análisis se soporta en base a unos objetivos generales y sobre estos unos más específicos que se desglosan a continuación para dotar de estructura al desarrollo de este modelo.

Los objetivos específicos a su vez se han escogido en base al enfoque y planteamiento elegidos para dar solución a los objetivos generales

Tras todas las decisiones tomadas se procede a la programación del modelo de datos y a su posterior análisis de los resultados obtenidos.

2. Objetivos

El objetivo del documento es el desarrollo de un modelo predictivo del riesgo de accidentes de tráfico viario en la ciudad de Madrid (España). Se pretende dar un valor de riesgo de accidente y gravedad esperada para cada zona de Madrid. El modelo se llevará a cabo a través de datos geoespaciales mediante técnicas de Deep Learning (DL). Se ha elegido como ya hemos justificado una Red Neuronal Profunda de Grafos (GCN) a la cual se le va a sumar una capa de memoria temporal (LSTM) buscando darle una mayor capacidad de predicción dentro del marco temporal del sistema, la cual será de utilidad debido a la alta estacionalidad del sistema de tráfico.

2.1. Objetivo general

Describir y desarrollar el modelo de red de manera que sea capaz de predecir tanto número de accidentes esperados como el tipo más probable de accidente para cada punto considerado por el mismo aplicado para el municipio de Madrid. Todo esto a través de datos accesibles desde diferentes fuentes y naturaleza relevantes para el caso

2.2. Planteamiento

2.2.1. Planteamiento de los datos

Para abordar una tarea de este tipo se precisan datos que sean influyentes en la causa de los accidentes, así como información de los mismos, del entorno y datos temporales y sociales que ubiquen los eventos en el tiempo.

- Datos relativos al riesgo de accidentes: Primordialmente se recogen datos históricos de la meteorología de la ciudad a través de diferentes plataformas que dan este tipo de servicios.
- Datos relativos al entorno: Se consultan datos de la red viaria de Madrid provistos por diferentes servicios de navegación y mapas. Esto dará tanto información de las vías en sí (tipo de vía, límite de velocidad, carriles, sentido...), comportamiento de la vía en cada momento (cantidad de coches, velocidad media), hitos cerca de las mismas (controles de velocidad, zona de hospitales, zona de colegios), también pudiera ser interesante información de eventos multitudinarios que pudieran afectar al volumen de personas y a la congestión del tráfico, véase eventos deportivos, musicales o populares.
- Datos temporales y sociales: Resultará útil saber el calendario laboral de la ciudad y del país para estimar los patrones de comportamientos de los vehículos.

- Datos de accidentes: Para estimar los accidentes pasados es necesario conocer cómo, dónde y cuándo se han producido los accidentes a lo largo del tiempo.

Los datos de los que se dispone son de diferentes fuentes que se unirán dentro de un mismo data frame para cada nodo del grafo, para lo cual hay que discretizar los datos repartidos por el marco temporal espacio para asignarle valores a los nodos concretos asignando de esta manera los accidentes cercanos a los nodos y también agregando los datos temporalmente en intervalos fijos para un tratamiento más uniforme.

2.2.2. Planteamiento del modelo

Tras analizar el contexto científico de los modelos de ciencia de datos relacionados con redes neuronales orientadas a la predicción de fenómenos espacio temporales (véase epígrafe 3, Estado del arte y Marco teórico), se ha optado por emplear una red neuronal basada en grafos con variables temporales y un uso de capas de células recurrentes como son las LSTM. Esto se logrará mediante la adición de células LSTM en una estructura mixta dentro del nodo.

Esta red es capaz de identificar patrones a nivel espacial a gran y pequeña escala por las sucesivas capas GCN y las dependencias temporales por sus capas LSTM.

La explicación detallada del modelo se desarrollará en el cuerpo principal del trabajo. Posteriormente, el modelo será implementado utilizando datos previamente procesados, y se obtendrán una serie de resultados que serán comparados con los obtenidos a partir de un modelo estándar.

2.3. Objetivos específicos

Para conseguir el objetivo general y alineado con las conclusiones que se extraen del planteamiento del problema se necesitan una serie de objetivos intermedios .

- Consecución de los datos y filtrado de las características más útiles.
- Desarrollo de la red del grafo y ajuste de los datos espaciales al mismo.
- Desarrollo del modelo predictivo utilizando las estructuras obtenidas.
- Entrenamiento, resultados y análisis.

3. Marco teórico y estado del Arte

3.1. Marco teórico

3.1.1. Redes neuronales

Las redes neuronales artificiales (ANN), y en especial las convolucionales (CNN) que son las más relevantes en los últimos años, están basadas en el concepto de neurona biológica y sus conexiones entre ellas, y son capaces de aprender a través de conjuntos de neuronas artificiales. Procesan y transmiten información, y resuelven de manera no algorítmica, problemas de clasificación y regresión entre otros.

Se estructuran en capas (layers); con una capa de entrada (input layer), capas ocultas (hidden layers) y una capa de salida (output layer). Las señales que ingresan en las neuronas tienen unos pesos y la suma de cada una dará unos valores totales a la siguiente capa, ajustando estos pesos debidamente, se llega a una solución. Este proceso ocurre de forma iterativa, y los pesos se ajustan a medida que pasan las iteraciones.

El proceso que ajusta los pesos de cada conexión se llama retropropagación o Backpropagation, iterando, con la intención de editar los pesos que minimizan el error producido a la salida para ir mejorando a cada paso a lo largo de muchas iteraciones. Este paso ha sido reconocido como la clave para encontrar patrones en problemas complejos y minimizar el error. (Dayhoff & DeLeo, 2020)

Aplicaciones: Las aplicaciones son tan diversas como el diagnóstico médico, el reconocimiento de voz o la detección de fraudes, con utilidad en el análisis económico y de comportamiento, entre muchos otros campos. (Crick, 1989)

3.1.2. Capas LSTM

Las LSTM fueron diseñadas inicialmente para gestionar adecuadamente la secuencia de datos en el tiempo y también para capturar una fuerte estructura o dependencia temporal a largo plazo, como en series de tiempo y procesamiento de lenguaje.

Estas capas están explícitamente diseñadas con unidades de memoria y tres puertas principales —la puerta de entrada, puerta de olvido y puerta de salida— que especifican qué información debe ser utilizada o desechada. Es un paso clave para evitar el desvanecimiento del gradiente y mantener la información útil.

Esto mismo, retener información en el tiempo hace que las redes sean muy adecuadas para predicción de series temporales, con aplicaciones en pronósticos meteorológicos, detección de anomalías financieras o análisis de sentimientos, en los que el contexto del pasado es clave. (Yao et al., 2015)

3.1.3. Redes de grafos

Las redes de grafos representan datos interconectados usando nodos y aristas, y encuentran aplicaciones en la modelización de relaciones espacio-temporales y relaciones geoespaciales. Como se ha explicado con anterioridad, esta modelización del espacio es mucho más óptima en información que la representación en imágenes u otro dato estructurado.

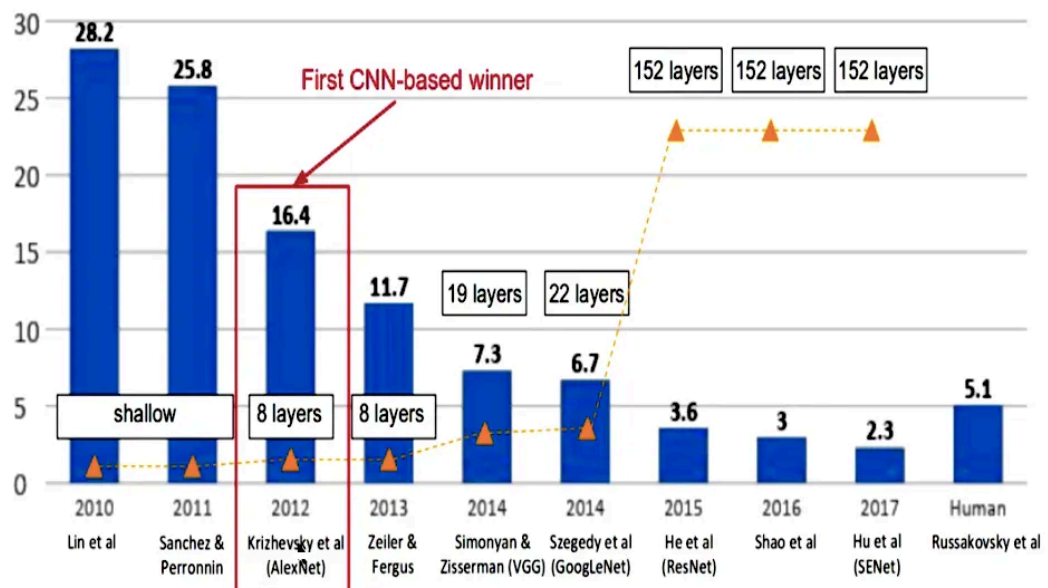
Grafos Neuronales (GNN): Los modelos de aprendizaje profundo son utilizados para la clasificación de nodos, la predicción de enlaces y la estructuración de datos complejos. Este enfoque particular ha sido eficaz en la predicción de eventos para movilidad y transporte con relaciones complejas y no tan evidentes. (Zhou, & Cui, 2018)

3.2. Estado del arte

El concepto teórico actual de las redes neuronales es amplio y tiene una larga historia, remontándose a 1943 con el modelo de red neuronal artificial desarrollado por Warren McCulloch y Walter Pitts. El Perceptrón, desarrollado por Frank Rosenblatt en 1958, fue un avance significativo en este campo. No obstante, estos conceptos no se popularizaron masivamente hasta varios años después debido a limitaciones en su concepción inicial y la falta de potencia computacional suficiente para su implementación efectiva.

No fue hasta 1986, con la introducción del concepto de backpropagation por David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams (J. Schmidhuber, 2022) , que solucionaron problemas relacionados con la no linealidad, lo que llevó a su popularización en el mundo de la computación. Este algoritmo permitió entrenar redes neuronales multicapa de manera más eficiente, resolviendo problemas que antes eran intratables.

En los años 90, los avances en redes neuronales, especialmente en el reconocimiento de imágenes, consolidaron su aplicabilidad en la ciencia. Ejemplos notables incluyen LeNet-5, desarrollado por Yann LeCun et al. para el reconocimiento de dígitos escritos a mano (Y. Lecun et al., 1998), y otras estructuras como las Redes Neuronales Recurrentes (RNN) y las Long Short-Term Memory (LSTM), introducidas por Sepp Hochreiter y Jürgen Schmidhuber en 1997. Estas redes ayudaron a resolver problemas significativos como el desvanecimiento del gradiente y la retención temporal, mejorando el rendimiento en tareas secuenciales (Kumar, 2021).

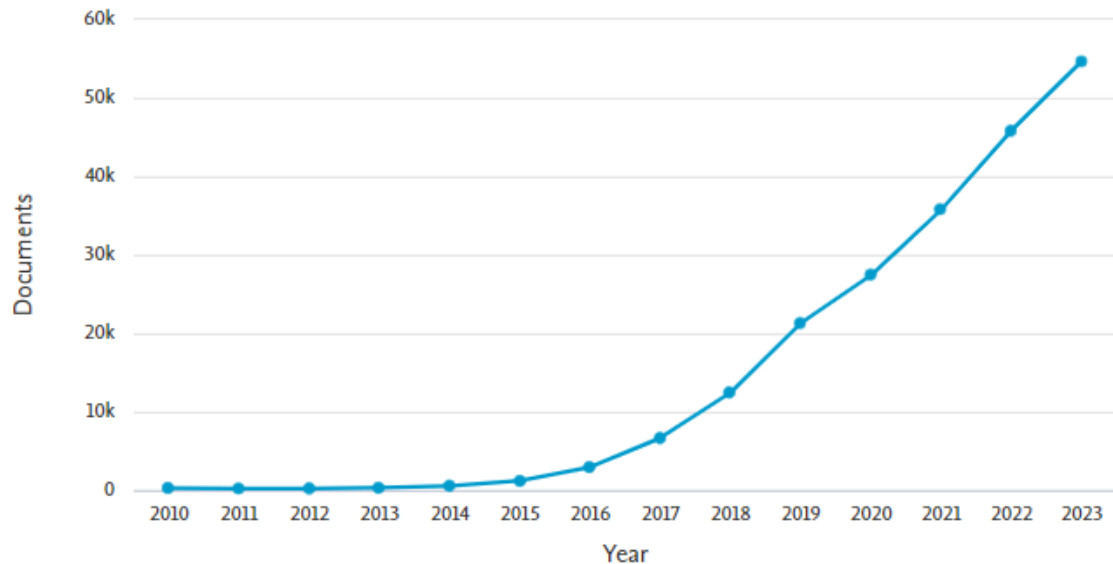


Las referencias y trabajos sobre aprendizaje profundo basado en redes neuronales han crecido exponencialmente en los últimos años, especialmente desde 2012 ((Krizhevsky et al., 2012,)). En ese año, AlexNet, desarrollado por Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton, se introdujo en el campeonato de ImageNet, logrando una reducción significativa en la tasa de error y marcando un hito en el campo de la visión por computadora. Este logro les valdría posteriormente el Premio Princesa de Asturias en 2022 ((Premio Princesa De Asturias De Investigación Científica Y Técnica 2022 - Geoffrey Hinton, Yann LeCun, Yoshua Bengio Y Demis Hassabis, 2022))

También ese año, Schmidhuber presenta varios trabajos relativos a las Redes neuronales recurrentes y árboles de decisión en el instituto IDSIA donde presenta ese mismo año varios trabajos (Ciresan et al., 2012) (Dan et al., 2013) muy relevantes en varios campos y ganando varios concursos de aprendizaje y clasificación con los conceptos introducidos en varios campos (Schmidhuber, 2013)

Este auge en la computación y el aprendizaje profundo, impulsado por los buenos resultados de estas estructuras y su versatilidad de aplicación, ha llevado a un crecimiento exponencial en la cantidad y variedad de campos en los que la comunidad científica aplica estas tecnologías. En 2023, el volumen de publicaciones anuales en las que se mencionan redes neuronales (54,652) es casi 200 veces mayor (+19,500%) que en el año en que AlexNet ganó el campeonato (277), según fuentes bibliográficas. Este crecimiento refleja la creciente importancia y aplicabilidad de las redes neuronales en diversas áreas, desde la visión por computadora y el procesamiento del lenguaje natural hasta la medicina y las finanzas.

Documents by year



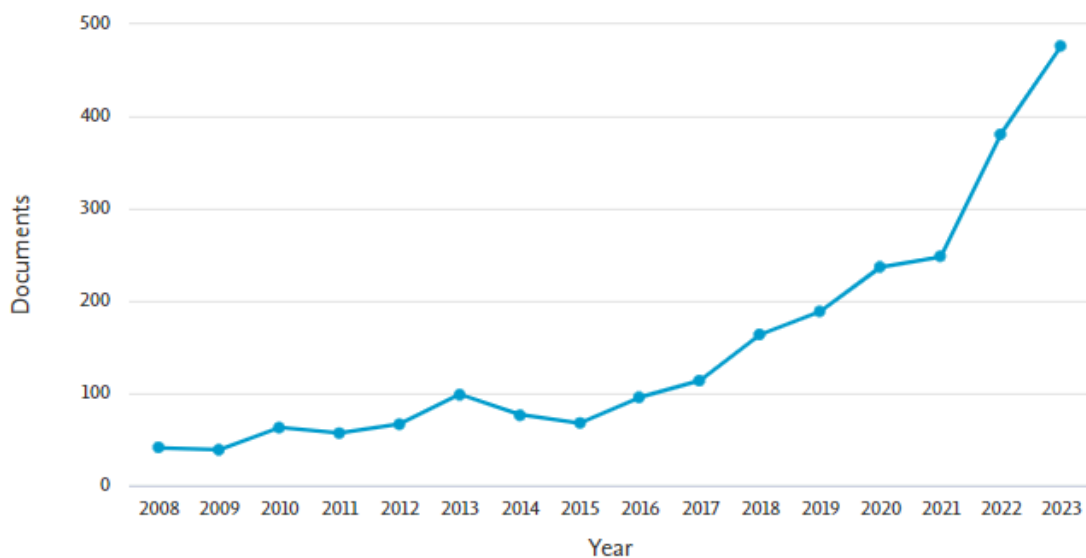
TITLE-ABS-KEY((cnn OR gnn OR dnn OR lstm)) AND PUBYEAR > 1989 AND PUBYEAR < 2026

210.212 document results

Select year range to analyze: 2010 to 2023 Analyze

Esto se ve reflejado en el campo en el que se encuentra este trabajo que también ha recibido grandes avances gracias a estos modelos, y los trabajos en los que intervienen redes neuronales para la predicción de los accidentes presenta una progresión similar aunque menos pronunciada (+~520%) desde el año de AlexNet (77) al último año completo, 2023 (476).

Documents by year



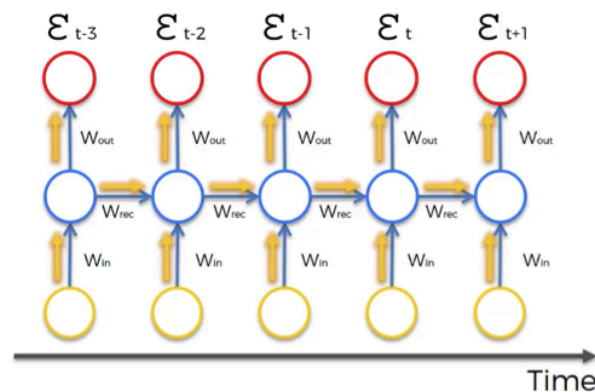
TITLE-ABS-KEY(road AND accident AND (forecast OR prediction)) AND (LIMIT-TO(SUBJAREA, "COMP") OR LIMIT-TO(SUBJAREA, "ENGI") OR LIMIT-TO(SUBJAREA, "MATH"))

2416 document results

Select year range to analyze: 2008 to 2023 Analyze

Dada la naturaleza temporal de las predicciones empiezan a cobrar importancia otro tipo de células, ya nombradas en este documento y desarrolladas años antes de su explosión, no obstante, es en este punto aproximadamente que empiezan a dar resultados y a usarse en publicaciones. Estas son capaces de captar mejor la capacidad temporal de los acontecimientos y con ello predecir de manera más precisa, dando paso a una nueva rama de las CNN, son las redes neuronales Recurrentes o RNN son capaces de aplicar la propagación gradiente a lo largo del tiempo., en contraposición con las CNN vistas hasta ahora, llamadas a partir de este momento FNN (Forward Neural Network, Redes neuronales de avance hacia adelante).

Las RNN no fueron muy populares ya que tenían ciertos problemas de propagación del gradiente (Desvanecimiento y explosión) que las hacían impredecibles.



(*Recurrent Neural Networks (RNN) - The Vanishing Gradient Problem - Blogs*, 2018) , no fue hasta la llegada de sus variantes LSTM o GRU son capaces de paliar este efecto, el desvanecimiento del gradiente, utilizando información a su entrada relativa a época pasadas y regulando flujo de información a través de las válvulas de memoria y de olvido a la de salida.

Aunque en estos trabajos se hicieron avances significativos durante los siguientes años para multitud de ámbitos científicos y aplicaciones basadas en imágenes, como para clasificación de imágenes, las redes neuronales, y en este caso el tipo convolucional, basado en transformaciones progresivas de imágenes. Por otro lado para otro tipo de campos basados en datos de naturaleza menos regular, y no en imágenes no funcionaban tan bien. Bien es cierto que las CNN cambiaron el paradigma y centraron la investigación en ese aspecto viendo su potencial gracias en

parte al desarrollo tecnológico de las Tarjetas gráficas modernas y la aceleración por Hardware (Strigl, 2010)

No obstante para otro tipo de tareas , no es el tipo de estructura de datos que se encuentra. Ya que los datos de otro tipo de estructuras pueden no ser regulares, para este tipo de problemas (Datos dispersos, Asociación de ideas/Sistemas de recomendación, relaciones irregulares, esquemas de datos), se diseñan y se popularizan, a partir de diferentes casos de éxito, (Google Deepmind, 2021,)

En los trabajos desarrollados como *Semi-supervised classification with graph convolutional networks* (Kipf & Welling, 2017) (*Graph Convolutional Neural Networks for Web-Scale Recommender Systems* Ying et al., 2018,) y posteriores revisiones sobre este tipo de redes, (*A Comprehensive Survey on Graph Neural Networks* Wu et al., 2019) *Graph Neural Network for Traffic Forecasting: A Survey* (Jiang & Luo, 2022, #), se evidencia cómo las redes de grafos parecen superar en tareas similares relativas al tráfico a cualquier otro planteamiento del momento con redes CNN convencionales.

Además en los trabajos relativos a las CNN para el tráfico se es capaz de predecir a corto plazo (Zhang et al., 2019) ya que otra de las faltas de la CNN es su poca retentiva temporal. Esto es solucionable por las CNN introduciendo las células Recurrentes RNN/LSTM y/o capas de atención (Jingyuan et al., 2016) , no obstante no se aproximan a los resultados de las GCN debido precisamente a la abstracción del mapa en los nodos y la bibliografía aunque también hay ejemplos de sistemas clásicos.

Esta dicotomía es tenida en cuenta y aplicada con éxito juntando ambas ideas *Diffusion Convolutional Recurrent Neural Network (DCRNN)*, (Li et al., 2018) y aplicando capas RNN a un autoencoder con GCN mejorando aún más los resultados.

Siguiendo con esta vertiente y colaborando con el gigante tecnológico Google, ingenieros de Deepmind lograron buenos resultados en la reducción del Tiempo estimado de Llegada de los trayectos en diferentes ciudades en *ETA Prediction with Graph Neural Networks in Google Maps*, (Google Deepmind, 2021) .

Ya entrando en la materia de accidentes de tráfico, una de las versiones presentadas en *Deep spatio-temporal graph convolutional network for traffic accident prediction* (Yu, 2021) con una vertiente más compleja que incluye también una solución espacio temporal para los grafos, mejora todos los modelos que el documento presenta a prueba.

Continuando el enfoque GCN, similar utilizando diferentes variantes de Redes neuronales con composición espacio-temporal encontramos trabajos con muy buenos resultados *T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction* (Zhao et al., 2015, #) Para este caso aparecen dos grandes tipos de modelos primordialmente temporales y primordialmente espaciales:

Del grupo espacial destacan las redes orientadas a grafos o GNN que dada su naturaleza se postulan dada la bibliografía (*A Comprehensive Survey on Graph Neural Networks* Wu et al., 2019) *Graph Neural Network for Traffic Forecasting: A Survey* (Jiang & Luo, 2022,). como la mejor alternativa para albergar dependencias espaciales, debidas a las restricciones espaciales más fuertes que tiene cada nodo con su vecino y a la propia naturaleza gráfica del sistema, no obstante, sólo son buenas prediciendo escenarios temporalmente locales, sin mucha desviación temporal de los datos.

En el apartado temporal en este contexto se aplican tanto métodos más clásicos como ARIMA O SARIMA dado el carácter periódico de los sistemas y que rivalizan con métodos más modernos de redes neuronales con capas recurrentes de memoria LSTM (Ren et al., 2017) que son capaces de retener información temporal lo cual dota de una variable muy importante.

Finalmente se han llegado a enfoques mixtos que aprovechan y combinan la buena característica de combinación de componente espacial que las GCN obtienen en este contexto, con otros modelos que permiten paliar sus defectos temporales como las LSTM como uno de los trabajos más relevantes podríamos citar SST_GCN *The sequential Based Spatio-Temporal Graph Convolutional Networks for minute-level and road-level traffic accident risk prediction* (Kim et al., 2024) con su modelo SST-GCN que aplica ambas capas con éxito de manera secuencial utilizando una capa de convolución de grafos con una capa temporal LSTM .

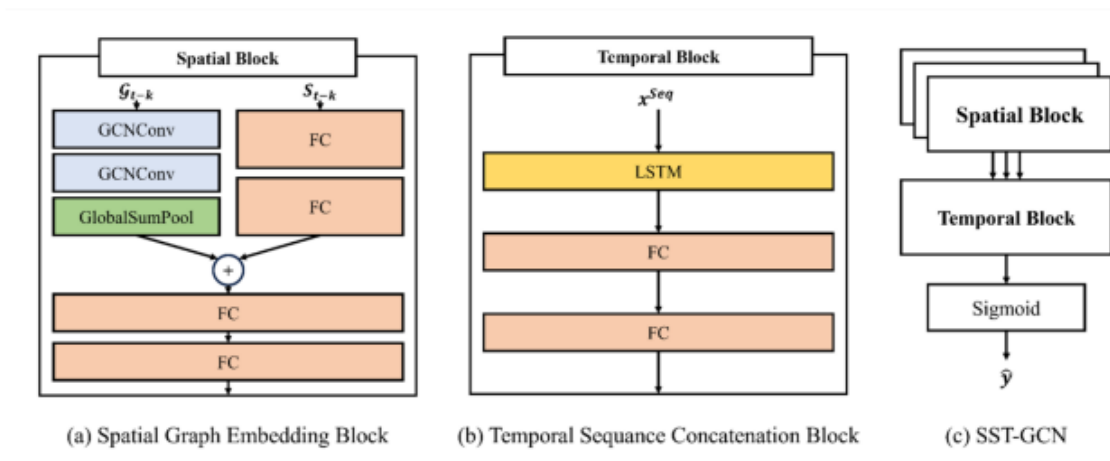


Figure 2: The Architecture of SST-GCN Model

Este tipo de redes mixtas con variante temporal mediante redes recurrentes (LSTM/GRU) y variante espacial (GCN), con diferentes adaptaciones y propuestas, son actualmente las dominantes para relaciones espacio temporales en las redes neuronales de grafos orientadas a predicción/ clasificación, como así lo indican las revisiones del tema más recientes (Luo et al., 2024) (Jin, 2024). Algunas de ellas

combinando los enfoques de manera consecutiva es decir primero una parte de convolución espacial, agregando la información de los nodos vecinos, y luego una capa temporal que transmita esa información en el tiempo. Este enfoque es más simple de programar y de parametrizar y ofrece algunas ventajas como su sencillez. La otra opción es la de entender la estructura Espacio-temporal como un todo y crear una estructura que engloba ambas dimensiones.

Otro enfoque a tener en cuenta con las redes de grafos es el de el embedding del grafo, que se trata de reducir la dimensión del grafo de manera que se engloban las informaciones de varios nodos de un nodo original para crear uno más fácil de tratar y que puede tener una información más agregada y menos dispersa. De esta manera los datos han de agregarse de manera que los datos se agrupen en espacio y tiempo en unos intervalos que tengan una densidad adecuada para el modelo.

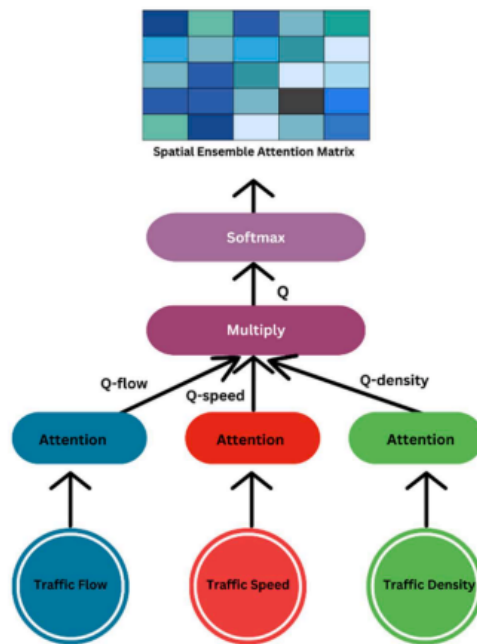
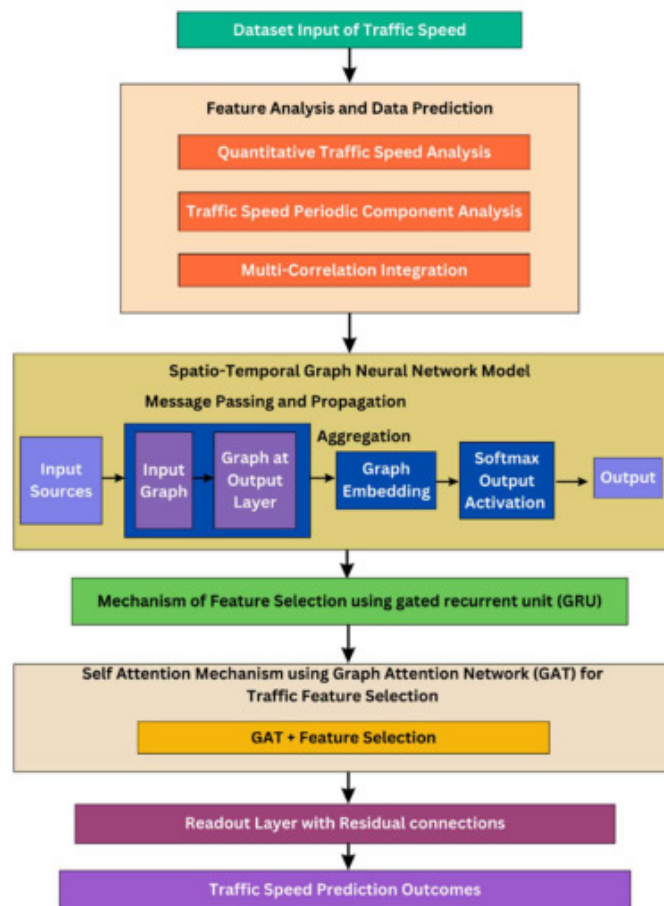


Fig. 2. Structure highlighting the spatial ensemble attention layer [54].

No obstante, además de la creación del modelo, parte crucial de la dificultad que entraña el nicho de los accidentes viales en concreto radica en las fuentes de información y cómo agrupar dicha información dentro del modelo. En este caso el nodo, al ser capaz de albergar información agregada de todo tipo es muy versátil para estas tareas de recogida de datos.



En este trabajo por ejemplo se observa como mezcla información y filtra mediante una capa de atención en cada fuente antes de mezclar en todo el modelo. (Shams Forruque, 202).

En este otro trabajo nos habla de todas las fuentes posibles, es desafío de agregar tal información tratarla mediante técnicas modernas de BIG DATA y retornar un modelo de DL de una manera eficiente con todo el proceso de datos que conlleva. (Li-Minn Ang & Kah Phooi Seng, n.d., 2022)

Los datos además pueden ser de diversa naturaleza y origen como se puede ver en esta revisión.

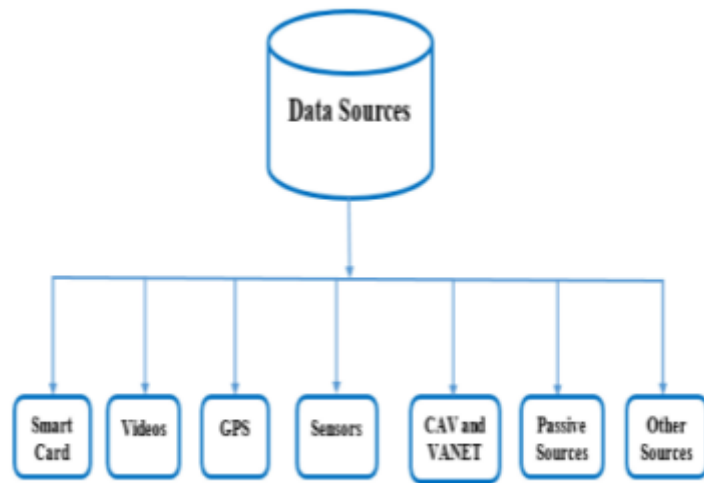
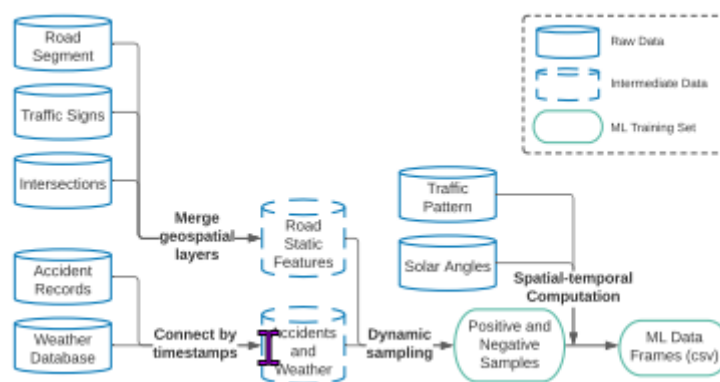


Figure 17. Data collection sources for SC transportation strategies.

Otros enfoques en este sentido lo que hacen es agrupar los datos por zonas geoespacialmente para asignar una localización común a todos los datos y así centralizar la información. (Ren et al., 2018) Este enfoque no se usa de esta manera en el documento pero es una práctica habitual dentro del tratamiento de grafos y de datos en general. agrupar en globos de datos tanto espacial como temporalmente. de esta manera podremos agregar datos de manera mucho más efectiva.

En otros modelos se adquiere la información de diferentes fuentes y dependiendo de su naturaleza se agregan de una u otra manera (Shi et al., 2021)



4. Desarrollo del proyecto y resultados

4.1. Enunciado formal del proyecto

El objetivo de este proyecto es desarrollar un dataset y su posterior modelado para implementar un sistema de predicción de accidentes de tráfico en la ciudad de Madrid. Para ello, se crea un modelo de red neuronal de convolución de grafos con una variable temporal. Además, se integra una capa de red neuronal LSTM en cada grafo, lo que permite predecir variables relativas a futuros accidentes, tales como:

- Tiempo hasta el siguiente accidente.
- Gravedad del siguiente accidente.
- Número de personas afectadas.

El modelo se alimenta de datos obtenidos de diversas bases de datos públicas y gratuitas, lo que permite abordar diferentes dimensiones del problema y obtener una representación más rica de los accidentes de tráfico.

4.2. Metodología

El proyecto se divide en varias etapas, cada una de las cuales corresponde a un paso clave en el proceso de desarrollo. A continuación, se detallan los pasos seguidos para la realización del trabajo.

- Elección de datos geoespaciales relevantes.
- Estudio de modelos relevantes en función de los datos de entrada.
- ETL Extracción transformación y Carga de los datos elegidos.
- Adaptación de datos para el modelo escogido.
- Desarrollo del grafo
- Desarrollo del modelo
- Resultados

Cada una de estas etapas se lleva a cabo utilizando el lenguaje de programación Python y varias librerías tanto propias como externas, todas ellas públicas y gratuitas. Las bibliotecas empleadas incluyen herramientas de ciencia de datos, análisis matemático y redes neuronales.

Librerías utilizadas

Entre las principales librerías usadas en el proyecto se encuentran:

Geopandas, Pandas, Numpy y Scipy: Para el tratamiento de datos y cálculos matemáticos en dataframes geoespaciales.

Matplotlib y Folium: Para la visualización de datos generales y geoespaciales.

Pytorch, Scikit-learn, Pytorch Geometric y NetworkX: Para el desarrollo del modelo de grafos y su clusterización.

Openmeteo: Para la obtención de datos meteorológicos mediante consultas a la API web.

4.2.1. Elección de datos geoespaciales relevantes

La selección de datos geoespaciales apropiados es crucial para el éxito del modelo predictivo. Se han identificado cuatro categorías principales de datos:

4.2.1.1. Datos de accidentes

Nuestra variable a predecir, la comunidad de Madrid ofrece una estadística completa de todos los accidentes registrados por la policía. Estos accidentes se reportan de manera anual. Y nos dan información como:

- Localización y hora del accidente.
- Calle y barrio donde ocurrió el evento.
- Número de expediente.
- Sexo de la persona involucrada.
- Presencia de alcohol o drogas.

4.2.1.2. Datos de tráfico

Gracias a la plataforma de datos del ayuntamiento de madrid tenemos acceso a una serie de registros históricos de tráfico mediante sensores repartidos por toda la ciudad. Estos sensores nos dan información periódica del volumen de tráfico, velocidad y otros factores, en total ofrece unos 4000 sensores, que reportan en diferentes periodos de de tiempo:

- Sensor
- Volumen de tráfico
- Velocidad media
- Densidad de vehículos

Se pondera previamente la realización de la búsqueda de una API para la extracción de datos de tráfico de diferentes plataformas de navegación GPS que aportan información a parte de diversas fuentes. Estos datos no los tienen de manera histórica, por defecto, lo ofrecen de manera instantánea, lo cual es un inconveniente para el marco de tiempo planteado para el proyecto, por lo que es inviable y se desecha la idea en favor de la anteriormente mencionada.

4.2.1.3. Datos meteorológicos.

Los datos meteorológicos tienen una gran influencia en la ocurrencia de accidentes de tráfico, por lo que son un componente vital del estudio. A través de la API de OpenMeteo se obtienen datos detallados que incluyen:

- Temperatura y humedad
- Precipitaciones
- Visibilidad
- Viento

Se particulariza además datos según la posición geoespacial de los nodos obtenidos lo cual aporta algo más de precisión. Estos datos además se ofrecen de manera horaria

4.2.1.4. Datos laborales y sociales

Otra de las variables condicionantes para el transporte de las personas es el uso de los vehículos, está influenciado por el calendario laboral, grandes eventos (eventos musicales, populares, deportivos), grandes acumuladores de gente (centros educativos, hospitales, centros deportivos) etc.

En el caso de nuestro modelo simplemente se ha tenido en cuenta el calendario laboral de la ciudad ya que no se ha encontrado una base de datos con la información suficiente para el resto. El calendario laboral indica:

- Si es día laboral
- Festividades
- Tipo de festividad (nacional, regional o local).

4.2.1.5. Fuentes de datos

Portal de datos abiertos del Ayto. Madrid

Gran parte de los datos son ofrecidos por el portal de datos abiertos del Ayuntamiento de Madrid (<https://datos.madrid.es>) para obtener:

- Registros históricos de accidentes
- Información tráfico sobre la red vial
- Datos del calendario laboral

API web OpenStreetMap

A través de diferentes agregadores de sistemas meteorológicos es posible hacer una serie de peticiones WEB mediante una API WEB, en mi caso se utiliza la plataforma OpenStreetMap que ofrece datos de fuentes públicas y fiables.

Se ha utilizado una API de servicios web para obtener datos en tiempo real y pronósticos. es una Web que incluye una API.

4.2.2. Estudio de modelos

El estudio de los modelos relevantes para este proyecto se realizó basándose en la documentación científica disponible y considerando los requisitos del sistema de predicción de accidentes. Los modelos evaluados se detallan en el apartado correspondiente del informe (véase el apartado 3).

4.2.3. ETL de los datos

Los datos utilizados provienen de diversas fuentes y formatos, y cada conjunto de datos posee diferentes estructuras temporales, nomenclaturas y referencias. Para garantizar la coherencia y funcionalidad del sistema, se realizó un proceso ETL (Extracción, Transformación y Carga) exhaustivo. Entre las tareas llevadas a cabo, se incluyen:

4.2.3.1. Datos de Tráfico.

Los sensores de tráfico distribuidos por la ciudad reportan datos en intervalos de 15 minutos. Estos datos se almacenan en archivos CSV mensuales que contienen las medidas de cada sensor. Para identificar cada sensor, se dispone de una tabla con su ID, coordenadas y la ubicación de la vía correspondiente.

Debido a la alta cantidad de sensores y la granularidad de los datos de accidentes, se decidió agregar la información de los sensores en clusters, utilizando su localización geoespacial para simplificar la predicción y reducir el coste computacional. Esto también permite que el modelo espacial sea compatible con un enfoque de grafos.

Además parte de los datos recogidos presentaban algún tipo de problema en la recogida y se emplean diferentes métodos de limpieza y rellenado para que el modelo sea completo y no contenga valores nulos.

Algunos sensores presentaban errores en la recogida de datos o cambios ligeros de localización. Para evitar que esto afecte al clustering y al modelo final, se aplicaron métodos de limpieza y se filtraron los sensores problemáticos.

4.2.3.2. Datos de accidentes.

Los datos anuales sobre accidentes proporcionan una gran cantidad de información, aunque su granularidad es baja debido a que solo se reportan varios miles de accidentes al año en toda la ciudad. Esto significa que los accidentes están infrarepresentados en comparación con otros datos del modelo, como los de tráfico. Para adaptar esta información al sistema, se llevaron a cabo varios pasos:

Referenciación espacial y temporal

Los datos de accidentes se presentan con una referencia espacial que indica el lugar exacto del accidente, así como la hora en la que ocurrió. Sin embargo, es necesario redondear y ajustar estos datos para que se alineen con la estructura temporal y espacial del grafo. Esto requiere procesos de agregación y un tratamiento detallado de la información.

Ajuste de datos al modelo

Los nodos del grafo representan la media de un cluster de sensores, por lo que los accidentes se asignan a estos clusters en función de su localización. Este proceso asegura que cada nodo tiene asignado un número coherente de accidentes en función de su posición y su relación temporal.

Desafíos en la agregación de datos

Dado que los accidentes tienen una granularidad mucho menor que los datos de tráfico, se debe tener cuidado al realizar la agregación. Se busca mantener la mayor precisión posible en las predicciones, sin perder datos relevantes ni introducir sesgos.

4.2.3.3. Datos meteorológicos.

Los datos meteorológicos, son recogidos mediante la API directamente y se agregan al Data Frame de Python, ya pormenorizados con la georeferencia por cada nodo de los clusters creados previamente para una mayor precisión. Estos datos son aportados de manera horaria y son considerados constantes para esa hora.

4.2.3.4. Datos Calendario

El calendario laboral de la ciudad de Madrid es otro factor importante a considerar, ya que los patrones de tráfico y el uso de vehículos están estrechamente relacionados con los días laborables, festivos y eventos especiales. Los datos del calendario incluyen información desde el año 2013 hasta 2024 y se han organizado de manera que puedan ser filtrados y codificados por fecha. El calendario laboral de la ciudad de Madrid es otro factor importante a considerar, ya que los patrones de tráfico y el uso de vehículos están estrechamente relacionados con los días laborables, festivos y eventos especiales. Los datos del calendario incluyen información desde el año 2013 hasta 2024 y se han organizado de manera que puedan ser filtrados y codificados por fecha.

Tras ver el volumen de datos se tendrá en cuenta un total de 20 meses. Al ser un dataset temporal se ha de tener en cuenta la distribución de los datos para no interferir en el modelo.

Por tanto los datos se recogerán de manera correlativa temporalmente, es decir seguidos en el tiempo. Se reparten en un subconjunto de 14 meses para entrenamiento, un 70% del tiempo. 4 meses de validación, 20% y 2 meses de predicción 10%. El periodo elegido es de Octubre de 2022 a Mayo del 2024

Además de todos estos, se añaden columnas de referencia temporal ya que los índices, temporales, se perderán al pasar el modelo a un sistema de tensores matemáticos y necesitar esto una referencia únicamente numérica y se separará la fecha en año mes día y hora (en formato decimal).

4.2.4. Desarrollo del grafo

La creación del grafo que representa la red vial de Madrid se realiza mediante los siguientes pasos:

1. Extracción de la topología total de los sensores.

La gran cantidad de datos de sensores hace complicado el tratamiento, análisis y agregación de los mismos por lo tanto es necesario dividir el sistema primeramente en estructuras de datos más pequeñas y usar solo lo necesario (ids únicos y sus coordenadas) para poder reducir los datos y manejar todos de golpe dentro del grafo

2. Asignación de los nodos del grafo a partir del clustering realizado a los sensores de medida del tráfico.

A cada ID de sensor se le asigna un clúster y los datos se agregan. Reseñar que todos los elementos han de estar comprendidos dentro de alguno de estos nodos del grafo.

3. Creación de aristas y del árbol de mínima expansión.

Los nodos del grafo se hallan a través de la media de los sensores que lo forman, y por otro lado las aristas tienen un valor igual a la distancia que separa cada nodo. Importante reseñar que el grafo a fin de aligerar el costo computacional se ha elegido una estructura de aristas de mínimo árbol de expansión, que quiere decir que en lugar de todas las aristas, únicamente unen dos nodos de modo que el grafo resultante tenga [MST] las menos conexiones posibles sin que quede un solo nodo huérfano. Esto extrae parte de la componente real de transición entre nodos, pero aligera el alto componente computacional que tienen los modelos de grafos y los LSTM.

4. Referencia temporal continua:

Dado que los datos se procesan en una estructura de grafo, es crucial que todos los nodos compartan una referencia temporal común. Para ello, se utiliza una referencia temporal continua (Range Datetime), asegurando que no falten datos y que las convoluciones mantengan su correlación temporal. Si es necesario, se interpolan algunas medidas para garantizar la coherencia de los datos.

5. Asignación de atributos

Finalmente, los atributos (meteorológicos, de accidentes, etc.) se asignan a los nodos y aristas en función de sus coordenadas geoespaciales. Esto se realiza utilizando la librería **NetworkX**, que permite crear una estructura de datos de grafo compatible con el modelo de predicción.

4.2.5. Adaptación de datos para el modelo escogido.

Los datos teniendo en cuenta todo lo anterior, uno de los mayores desafíos es transformar los datos y adaptarlos de manera que cumplan con todos los requisitos y teniendo en cuenta todas las restricciones impuestas por el modelo a tratar:

- **Selección de características y filtrado de valores**

Los datos brutos tienen columnas que nos son de interés para el modelo, redundantes o que meten ruido o imprecisiones, por ello es importante controlar dichas variables antes, durante y tras el proceso de limpieza.

- **Redes neuronales**

El modelo elegido es una red neuronal que trabaja con datos numéricos, por lo que es necesario convertir ciertas columnas de datos en formatos compatibles. Además, los modelos neuronales se benefician de trabajar con rangos de datos centrados alrededor de 0, por lo que se escaló y normalizó la información.

Además la estructura de datos final tiene que ser de manera que sea compatible con Pytorch Geometric

- **Desbalanceo y granularidad**

Dado que los accidentes de tráfico son eventos relativamente raros en comparación con la cantidad de datos de no-accidente, el modelo debe ser adaptado para manejar esta asimetría de clases. Se ha optado por un submuestreo aleatorio de las medidas sin accidente, con el objetivo de mantener una proporción 1:1 entre las clases de accidente y no-accidente.

- **Discretización de variables continuas:**

Variables continuas con valores grandes difícilmente escalables, como el tiempo a un accidente, que puede ser muy alto pero si se escala los valores bajos pierden peso, se tendrán que discretizar o truncar para evitar este efecto

- **Grafo**

Por el hecho de ser un grafo como hemos dicho, la información se estructura alrededor de los nodos y aristas, por lo tanto cada evento ocurrido en el espacio (accidentes, medidas de sensores, ocurrencia de otros eventos) debe tener un nodo asociado.

Además los accidentes no se dan igual en todos los nodos luego cada nodo tiene unos eventos diferentes a un paso de tiempo diferente.

- **Complementarios**

Además se debe hacer un estudio de variables y estudiar cual se puede eliminar en pos de aligerar complejidad de cómputo al modelo, así como otros ajustes, como agregar columnas extra para el tratamiento del modelo como puede ser una columna que indica el enmascaramiento de las medidas para el desbalanceo.

4.2.6. Justificación del modelo

El proyecto se justifica por la necesidad de modelar y predecir accidentes de tráfico en la ciudad de Madrid, un fenómeno que depende de múltiples factores temporales y geoespaciales, lo que hace que los métodos tradicionales de predicción (como regresiones simples) no sean suficientes. Como se ha discutido previamente, estos modelos se han demostrado muy capaces para este tipo de áreas espaciales y/o temporales

Redes Neuronales de Grafos (GCN):

- Representación Espacial Compleja. Los datos de tráfico en Madrid están distribuidos espacialmente a lo largo de una red vial compleja, por lo que se

necesita un modelo que capture estas relaciones. Los modelos de grafos son ideales para representar redes viales.

- Captura de Dependencias Espaciales. Los GCN permiten modelar cómo los cambios en una parte de la ciudad (un nodo del grafo) afectan a las partes cercanas. Esto es crucial en la predicción de accidentes, ya que un aumento del tráfico en una zona puede influir en zonas adyacentes.
- Eficiencia Computacional: Aunque es costoso computacionalmente tratar todos los nodos y aristas, se ha utilizado un **árbol de mínima expansión (MST)** para simplificar las conexiones entre nodos y hacer el procesamiento más manejable sin perder demasiada precisión.

Capas LSTM (Long Short-Term Memory)

- Captura de Dependencias Temporales. Las capas LSTM permiten capturar relaciones a largo plazo en los datos temporales. Dado que los accidentes de tráfico no son eventos puntuales sino que dependen de patrones repetitivos a lo largo del tiempo (como horas pico, condiciones climáticas), es esencial un modelo que pueda recordar eventos pasados y hacer uso de ellos en la predicción de eventos futuros
- Eficiencia en la modelización temporal. Las LSTM están diseñadas para manejar secuencias de datos, lo que es ideal para series temporales como las de tráfico o accidentes. Permiten que el modelo aprenda la influencia de eventos pasados (por ejemplo, un pico de tráfico dos horas antes) sobre la probabilidad de un accidente.

Se ha optado por una combinación de Redes Neuronales de Convolución de Grafos (GCN) y capas LSTM debido a las características específicas de los datos:

Otros enfoques considerados fueron.

GCN (Graph Convolutional Network) simple

Este modelo fue desechado. Debido a la simplicidad. Y a la falta de tratamiento de los datos temporales. Se implementa como una parte del modelo presentado y es la base para esta rama de modelos.

GAT (Graph Attention Network)

Se trata de una mejora de la anterior. Pero aún así, pone el foco en la variable espacial. La variable temporal es especialmente relevante en este tipo de eventos.

DCRNN (Diffusion Convolutional Recurrent Neural Network)

Modelo más complejo y eficiente, pero también mucho más costoso de implementar. Este modelo requiere mucho tratamiento de los datos, los cuales no se adecuaban a nuestro set de datos.

Nuestro modelo presenta un equilibrio entre la parte temporal y espacial así como un tratamiento integral del problema del desbalance de clases, filtrando internamente los eventos relevantes durante el entrenamiento.

Se han de tomar una serie de decisiones más concretas relativas al modelo las cuales se detallan en el siguiente epígrafe

4.3. Desarrollo del proyecto

Como ya se ha indicado, se tienen 3 bases de datos relativas a la entrada. Meteorología, Sensores de Tráfico y Calendario laboral. Se presentan las variables tratadas, aunque después hayan sido depuradas en el proceso. Las no usadas se pueden consultar en los documentos anexos

Variable Sensores Tráfico	Significado
id	Identificación del punto de medida. Es de tipo secuencial, además de ser único e invariable.
fecha	Fecha y hora oficiales de Madrid con formato dd/mm/yyyy hh:mi:ss
intensidad	Número de vehículos en el periodo de 15 minutos, expresada en vehículos/hora. Un valor negativo implica la ausencia de datos.
carga	Parámetro de carga del vial en el periodo de 15 minutos. Representa una estimación del grado de congestión
ocupacion	Porcentaje de tiempo que está un detector de tráfico ocupado por un vehículo.
error	Indicación de si ha habido al menos una muestra errónea o sustituida en el periodo de 15 minutos.
vmed	Velocidad media de los vehículos en el periodo de 15min (Km./h).

Variable Ubicaciones Sensores	Significado
tipo_elem	Tipo de elemento: URB urbano, M30 situado en la M30, OTHER otro tipo
id	Identificador único numérico del punto de medida
longitud	Longitud en grados en WGS 84
latitud	Latitud en grados en WGS 84

Variable Calendario	Significado
Fecha	Día del año de forma consecutiva en formato dd/mm/aaaa
Dia_semana	Día de la semana correspondiente a la Fecha de columna 1
Laborable/festivo/domingo festivo	Para el día del año y el día de la semana señalados en las columnas 1 y 2 se indicará si es laborable, festivo o domingo que coincida con día festivo.
Tipo de festivo	Se indicará si se trata de festivo nacional, de la Comunidad de Madrid o local (ciudad de Madrid)
Festividad	Nombre dado al día festivo por la legislación de nuestro país

Y por otro lado 1 con los datos de salida relativos a los accidentes.

LESIVIDAD:

01 Atención en urgencias sin posterior ingreso. - LEVE

02 Ingreso inferior o igual a 24 horas - LEVE

03 Ingreso superior a 24 horas. - GRAVE

04 Fallecido 24 horas - FALLECIDO

05 Asistencia sanitaria ambulatoria con posterioridad - LEVE

06 Asistencia sanitaria inmediata en centro de salud o mutua - LEVE

07 Asistencia sanitaria sólo en el lugar del accidente - LEVE

14 Sin asistencia sanitaria

77 Se desconoce

En blanco Sin asistencia sanitaria

El resto de columnas no usadas, se descartaron, o por ser redundantes, o poco interesantes a priori, pues la carga computacional hacía imposible el tratamiento, o no ser el objetivo del trabajo.

Variable Accidentes	Significado
num_expediente	AAAASNNNNNN, donde: • AAAA es el año del accidente. • S cuando se trata de un expediente con accidente.
fecha	Fecha en formato dd/mm/aaaa
hora	La hora se establece en rangos horarios de 1 hora
cod_lesividad*	cod_lesividad*: Viene tipificado más abajo
coordenada_x_utm	ubicación coordenada x
Coordenada_y_utm	ubicación coordenada y

4.3.1. Desarrollo del proyecto

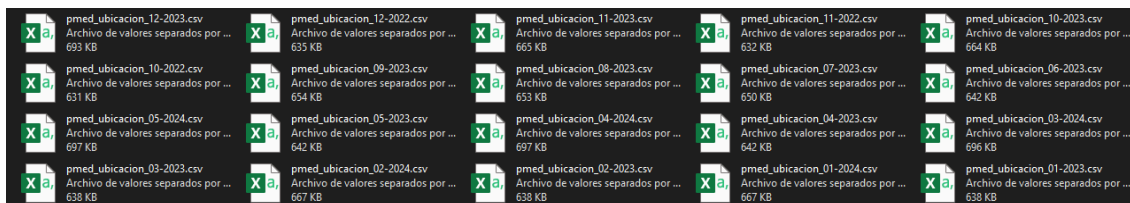
Para el desarrollo de este trabajo. se han llevado a cabo las siguientes subtarear con los datos anteriormente descritos. Aunque no se llega a todos los pasos de manera secuencial porque para cada uno de los pasos intermedios hacen falta pasos previos de otros apartados, las partes más importantes serían

- ETL de los datos
- Desarrollo del dataset espacio temporal completo
- Desarrollo del modelo de predicción
- Entrenamiento y resultados

4.3.1.1. Extracción, Transformación y Carga de los datos

Los datos al ser extraídos deben ser pretratados en función de su naturaleza y su formato y adecuarlos a las características de las estructuras de las que son objetivo, como son los grafos, tensores y las redes neuronales además del lenguaje de programación usado.

- Extracción de los puntos espaciales de ubicación de los sensores.
 - Los datos provienen de archivos csv distribuidos por meses.



- Además el formato no es el correcto para la correcta importación de datos. Por lo tanto se limpian para agregarlos a un dataframe.

```
os.chdir(base_folder + '/' + DS_folder_ubi)
print(os.getcwd())
for file in os.listdir():
    if file.startswith('pmed_'):
        sens[file] = pd.read_csv(file, sep = ';', usecols = ubi + ['id'])
        sens[file].set_index(keys='id', inplace = True)
        if sens[file]['latitud'].dtype == 'object':
            sens[file]['latitud'] = (sens[file]['latitud']
                                     .str.replace('.', '', regex = False)
                                     .str.slice(0, 10)
                                     .astype(float) / 10 ** 8)
            sens[file]['longitud'] = (sens[file]['longitud']
                                     .str.replace('.', '', regex = False)
                                     .str.slice(0, 10)
                                     .astype(float) / 10 ** 8)
        # print(sens[file][['longitud', 'latitud']].head(1))
        print('___PUNTOS_MEDICION_POR_MES___')

sens_ = pd.concat(sens.values())
```

- Por cada id de sensor le corresponde una única ubicación y en cada mes se repiten los sensores, por lo tanto eliminamos las duplicidades.

```
sens_ = pd.concat(sens.values())

sens_unico = sens_.reset_index().drop_duplicates(subset='id', keep='first').set_index('id')
```

- Extracción de los puntos temporales relevantes a los accidentes y medidas para filtrar datos.

```
medidas, ids = extraer_medidas(DS_folder_sens, med_file, cols_v)

df = sens_unico.loc[np.intersect1d(sens_unico.index, list(ids))]
```

- Extracción y transformación de las medidas e introducción en el grafo según su nodo.

```
for a, file in enumerate(os.listdir(folder) ):
    print(f'{a + 1}/{n}')
    if file.endswith('.csv'):
        medida = pd.read_csv(os.path.join(folder, file),
                               sep = ';',
                               usecols = cols_v,
                               encoding_errors='replace')
        medida['fecha'] = pd.to_datetime(medida['fecha'])
        # medidas[file]=medida[~medida['id'].isin(bad_ids)]
        medidas[file]=medida
        ids.update(medida.id)
```

```
for name, data in medidas.items():
    cluster_dates = data.groupby('cluster')['fecha'].apply(set)
    for cluster, dates in cluster_dates.items():
        if cluster not in fechas_en_nodo:
            fechas_en_nodo[cluster] = pd.DatetimeIndex([])
        fechas_en_nodo[cluster] = fechas_en_nodo[cluster].union(pd.DatetimeIndex(dates))
```

- Extracción y transformación de los datos de accidentes y adecuación al marco temporal.
 - Se extraen los accidentes y se transforman las columnas según la documentación del CSV y las características de interés.

```
Acc['fecha'] = Acc['fecha'] + ' ' + Acc['hora']
col_drop = ['localizacion', 'numero', 'cod_distrito', 'distrito', 'tipo_accidente',
            'tipo_vehiculo', 'tipo_persona', 'lesividad', 'positiva_alcohol',
            'positiva_droga', 'hora', 'sexo']
col_drop += [c for c in Acc.columns if c.startswith('estado_meteorol')]
print(col_drop)
Acc.drop(col_drop, axis = 1, inplace = True)

Acc.loc[(Acc['cod_lesividad'].isnull()) | (Acc['cod_lesividad'] == 77), 'cod_lesividad'] = 14
Acc.fecha = pd.to_datetime(Acc.fecha, format = '%d/%m/%Y %H:%M:%S')

# print('Accidentes antes', Acc)
Acc['coordenada_x_utm'].dropna(inplace = True)
Acc['coordenada_y_utm'].dropna(inplace = True)
Acc=Acc.loc[(Acc['coordenada_x_utm']!= '#¡VALOR!') | (Acc['coordenada_y_utm']!= '#¡VALOR!')]
Acc.dropna(inplace = True)
# print('Accidentes despues', Acc)

grupo_edad = {
    'De 0 a 17 años': ['Menor de 5 años', 'De 6 a 9 años', 'De 10 a 14 años', 'De 15 a 17 años'],
    'De 18 a 30 años': ['De 18 a 20 años', 'De 21 a 24 años', 'De 25 a 29 años'],
    'De 31 a 45 años': ['De 30 a 34 años', 'De 35 a 39 años', 'De 40 a 44 años'],
    'De 46 a 60 años': ['De 45 a 49 años', 'De 50 a 54 años', 'De 55 a 59 años'],
    'Más mayores de 60': ['De 60 a 64 años', 'De 65 a 69 años', 'De 70 a 74 años', 'Más de 74 años'],
    'Desconocido': ['Desconocido']
}
```

- Extracción y transformación del calendario laboral.

```
Cal = pd.read_csv (file, sep = ';', index_col = 0).iloc[:, :-2]
Cal.index = pd.to_datetime (Cal.index, format = '%d/%m/%Y')
Cal.loc [Cal ['Dia_semana'] == 'miércoles', 'Dia_semana'] = 'miercoles'
Cal.loc [Cal.Dia_semana == 'sábado', 'Dia_semana'] = 'sabado'
Cal.loc [Cal.laborable_festivo == 'Festivo', 'laborable_festivo'] = 'festivo'
Cal.loc [Cal.TipoFestivo == 'Fiesta de la Comunidad de Madrid', 'TipoFestivo'] = 'Festivo de la Comunidad de Madrid'
Cal.loc [
    Cal.TipoFestivo == 'Festivo de la comunidad de Madrid', 'TipoFestivo'] = 'Festivo de la Comunidad de Madrid'
Cal.loc [Cal.TipoFestivo == 'Fiesta Comunidad de Madrid', 'TipoFestivo'] = 'Festivo de la Comunidad de Madrid'
Cal.loc [
    Cal.TipoFestivo == 'traslado de la Fiesta de la Comunidad de Madrid', 'TipoFestivo'] = ('Festivo de la '
                                                                                          'Comunidad de Madrid')
Cal.loc [Cal.TipoFestivo == 'Fiesta local', 'TipoFestivo'] = 'Festivo local'
Cal.loc [Cal.TipoFestivo == 'Festivo local de la ciudad de Madrid', 'TipoFestivo'] = 'Festivo local'
Cal.loc [Cal.TipoFestivo == 'Fiesta nacional', 'TipoFestivo'] = 'Festivo nacional'

to_OH=['laborable_festivo','TipoFestivo']
to_Label=['Dia_semana']

Cal.reset_index (inplace = True)
Cal ['day'] = Cal ['Dia'].dt.day
Cal ['month'] = Cal ['Dia'].dt.month
Cal ['year'] = Cal ['Dia'].dt.year
LE=LabelEncoder()
for col in to_Label:
    Cal[col]=LE.fit_transform (Cal [col])
Cal = cyclic_encoder(Cal, col: 'Dia_semana')
Cal = pd.get_dummies(Cal, columns = to_OH)

to_drop=['Festividad', 'Dia']
Cal.drop (columns = to_drop, inplace = True)
```

- Consultas API y ETL para los datos meteorológicos según los datos espaciales de los nodos del grafo.
 - Se extraen las peticiones para cada nodo por lotes de datos según el máximo permitido por la API.

```
cache_session = requests_cache.CachedSession ( cache_names='cache', expire_after = -1)
retry_session = retry (cache_session, retries = 5, backoff_factor = 0.2)
openmeteo = openmeteo_requests.Client (session = retry_session)

url = 'https://archive-api.open-meteo.com/v1/archive'
all_results = []
batches = list (chunker (Coords.values.tolist (), size=10))
datemin = pd.to_datetime (datemin) - pd.Timedelta (days = 1)
datemax = pd.to_datetime (datemax) + pd.Timedelta (days = 2)

print ('-----')
print (f'Desde {datemin.strftime ('%Y-%m-%d')} -- Hasta {datemax.strftime ('%Y-%m-%d')}')
for i, batch in enumerate (batches):
    try:
        print (f'Batch {i + 1} de {len (batches)}')

        params = {
            'latitude': batch [0],
            'longitude': batch [1],
            'start_date': datemin.strftime ('%Y-%m-%d'),
            'end_date': datemax.strftime ('%Y-%m-%d'),
            'hourly': ['temperature_2m', 'relative_humidity_2m', 'dew_point_2m', 'precipitation', 'rain', 'is_day',
                       'sunshine_duration'],
            'timezone': 'Europe/Berlin',
        }
        all_results.extend (openmeteo.weather_api (url, params = params))
```

- Se añaden los datos al Data Frame y se integran en el grafo según su nodo

```
hourly_dict = {}

for nodo, response in enumerate (Responses):
    # print(f'CL {nodo}: {np.round(response.Latitude(),5) }°N {np.round(response.Longitude(),5)}°E')

    hourly = response.Hourly ()
    hourly_temperature_2m = hourly.Variables (0).ValuesAsNumpy ()
    hourly_relative_humidity_2m = hourly.Variables (1).ValuesAsNumpy ()
    hourly_dew_point_2m = hourly.Variables (2).ValuesAsNumpy ()
    hourly_precipitation = hourly.Variables (3).ValuesAsNumpy ()
    hourly_rain = hourly.Variables (4).ValuesAsNumpy ()
    hourly_is_day = hourly.Variables (5).ValuesAsNumpy ()
    hourly_sunshine_duration = hourly.Variables (6).ValuesAsNumpy ()

    hourly_batch = {'date': pd.date_range (
        start = pd.to_datetime (hourly.Time (), unit = 's', utc = True),
        end = pd.to_datetime (hourly.TimeEnd (), unit = 's', utc = True),
        freq = pd.Timedelta (seconds = hourly.Interval ()),
        inclusive = 'left'
    )}

    hourly_batch ['temperature_2m'] = hourly_temperature_2m
    hourly_batch ['relative_humidity_2m'] = hourly_relative_humidity_2m
    hourly_batch ['dew_point_2m'] = hourly_dew_point_2m
    hourly_batch ['precipitation'] = hourly_precipitation
    hourly_batch ['rain'] = hourly_rain
    hourly_batch ['is_day'] = hourly_is_day
    hourly_batch ['sunshine_duration'] = hourly_sunshine_duration

    hourly_batch = pd.DataFrame (hourly_batch)

    hourly_batch ['date'] = pd.to_datetime (hourly_batch ['date'])
    hourly_batch ['year'] = hourly_batch ['date'].dt.year
    hourly_batch ['month'] = hourly_batch ['date'].dt.month
    hourly_batch ['day'] = hourly_batch ['date'].dt.day
    hourly_batch ['hour'] = hourly_batch ['date'].dt.hour
```

```
meteo_data = meteo(Centers, datemin, datemax)
print(meteo_data[0].head())
G = Merge_Graph(G, meteo_data, on = ['year', 'month', 'day', 'hour'], how = 'inner', setindex = True)
```

4.3.1.2. Desarrollo del dataset

Todos los datos ya limpios es necesario ordenarlos y conjugarlos dentro de un mismo sistema temporal. Para ello se lleva a cabo. Un proceso que consta de los siguientes pasos.

- Clustering de los puntos para formar un grafo geoespacial mediante Kmeans.

```
COLXY = ['longitud', 'latitud']
print('CLUSTERING...')
kmeans = KMeans(n_clusters = clusters, random_state = 106)
clusters = kmeans.fit_predict(df_)

df_['cluster'] = clusters.astype(int)
cluster_centers = kmeans.cluster_centers_

# df con las coordenadas de los centros
Centers = pd.DataFrame(cluster_centers[:, 2:4], columns = COLXY)
df_cc = pd.DataFrame(cluster_centers, columns = cols)
```

- Extracción de las características geoespaciales del grafo según el clustering y conexionado del mismo.
 - A través de las coordenadas geoespaciales de los datos, se conectan mediante un árbol de mínima expansión. con la distancia como valor de referencia

```
print ('NO hay grafo en ', os.getcwd ())
positions = df.values
dist_matrix = distance_matrix (positions, positions)

G = nx.Graph ()

G.add_nodes_from (df.index)

for i, node1 in enumerate (df.index):
    for j, node2 in enumerate (df.index):
        if i < j:
            G.add_edge (node1, node2, weight = dist_matrix [i, j])

mst = nx.minimum_spanning_tree (G)
```

- Asignación de nodos a las medidas según la ubicación del sensor y su cluster asignado.

```
for name, data in medidas.items():
    # medidas[name]['cluster'] = data.id.map(df.cluster.to_dict())
    data['fecha'] = pd.to_datetime(data['fecha'])
    data['cluster'] = data.id.map(df.cluster.to_dict())
```


- Escalado regular y filtrado de las medidas de los sensores según los accidentes.
 - Se obtienen los pasos de tiempo en los que existen accidentes, y los elementos anteriores.

```
print('Se sacan los timesteps unicos de los accidentes...\n')
fechas_solo_acc = merge_acc.index.unique().sort_values()

print('Se añaden los pasos anteriores de accidentes, los no accidentes se han de filtrar por nodo...\n\n')
filtro_solo_acc = pd.DatetimeIndex([])
last = fechas_solo_acc

for p in range(1, pasos + 1):
    last = last - pd.Timedelta(minutes = 15)
    filtro_solo_acc = filtro_solo_acc.union(last)

del last
fechas_no_acc = {}
print('Teniendo las fechas sin accidentes (casi todas), '
      'se eligen algunas para tener en cuenta al modelo para ese nodo.\n\n'
      'El numero depende del nodo ')

```

- Así como el mismo número de puntos donde no hay accidentes.

```
print('Teniendo las fechas sin accidentes (casi todas), '
      'se eligen algunas para tener en cuenta al modelo para ese nodo.\n\n'
      'El numero depende del nodo ')
for nodo, f in fechas_en_nodo.items():
    # print(nodo, ' ### ', f.size)
    ##
    todo_no_acc = (~f.isin(fechas_solo_acc))
    n_acc = (merge_acc['cluster'] == nodo).sum()
    np.random.seed(106)
    fechas_no_acc[nodo] = pd.DatetimeIndex(np.random.choice(f[todo_no_acc],
                                                            n_acc, replace = False))

```

- De esta manera se obtiene un filtro para las medidas y así solo obtener medidas relevantes.

```
filtro_no_acc = {}
for nodo, f in fechas_no_acc.items():
    index = pd.DatetimeIndex([])
    last = f
    for p in range(1, pasos + 1):
        last = last - pd.Timedelta(minutes = 15)
        index = index.union(last)
    filtro_no_acc[nodo] = index

## Se añaden a los accidentes los no accidentes elegidos en los nodos,
# pues se necesitarán para el LSTM
filtro_total = filtro_solo_acc
for nodo, f in filtro_no_acc.items():
    filtro_total = filtro_total.union(f)

print('FILTRO TOTAL DE FECHAS CON ACCIDENTE (', fechas_solo_acc.size, ') Y ', pasos, ' ANTERIORES: ', filtro_total.size)
return acc_cluster, filtro_total

```

```
range_dates = range_dates.intersection(filtro_total)
```

- Introducción de medidas de meteorología en el grafo espaciotemporal según el nodo de manera horaria.

```
meteo_data = meteo(Centers, datemin, datemax)
print(meteo_data[0].head())
G = Merge_Graph(G, meteo_data, on = ['year', 'month', 'day', 'hour'], how = 'inner', setindex = True)
print(G.nodes[0]['medidas'].head())
print(G.nodes[0]['medidas'].shape)
```

- Introducción del calendario laboral, completando los datos de entrada.

```
to_drop=['Festividad','Dia']
Cal.drop (columns = to_drop, inplace = True)

print (Cal.head (2))

return Merge_Graph (G, Cal, on = ['year', 'month', 'day'], how = 'left', node_dependency = False,
                    setindex = True, drop_col = [])
```

- Asignación de nodos a los accidentes con técnicas geoespaciales.
 - Se asignan nodos a los accidentes por proximidad espacial a los mismos.

```
tree = cKDTree (np.array (list (zip (Coords.longitud, Coords.latitud))))
distancias, indices = tree.query (np.array (list (zip (gdf.geometry.x, gdf.geometry.y))), k = 1)
gdf ['cluster'] = indices

drop_col = ['coordenada_x_utm', 'coordenada_y_utm']
gdf.drop (drop_col, axis = 1, inplace = True)
```

- Inclusión de los accidentes en los nodos y tiempos correspondientes, adecuando las columnas para el modelo.
 - Una vez tienen nodo asignado, los accidentes unificando los índices temporales, se adhieren al grafo de los datos de entrada completando los datos del dataset. Además se añaden en las filas donde hay accidente unas columnas auxiliares para la transformación de variables de salida en función del siguiente accidente.

```
dic = 'medidas'
indexes = []
acccols = acc_cluster[0].columns.drop(['fecha', 'pad', 'hay_accidente'])
print("transform_dataset")
equal_size_graph(6)
for nodo, dict_ in 6.nodes.items():

    dict_[dic]['hour'] = dict_[dic]['hour'] + dict_[dic]['minute'] / 60 ## Decenas de minutos
    dict_[dic].drop('minute', axis = 1, inplace = True)
    to_merge = dict_[dic].reset_index()

    6.nodes[nodo][dic] = to_merge.merge(acc_cluster[nodo],
                                      on = 'fecha',
                                      how = 'left')

for nodo, dict_ in 6.nodes.items():

    dict_[dic][['pad', 'hay_accidente']] = dict_[dic][['pad', 'hay_accidente']].fillna(False).infer_objects(copy=False)
    dict_[dic].loc[:, acccols] = dict_[dic].loc[:, acccols].fillna(0).infer_objects(copy=False)
    6.nodes[nodo][dic] = prepare_next_acc(dict_[dic], acc_cluster[nodo].columns.drop(['hay_accidente', 'pad', 'fecha']))
    # print('-----')
```

- También se transforma la salida de los accidentes en función del siguiente accidente. y se completan los datos de manera dinámica.

```
next_c = [d + '_next' for d in to_next]

df = df.drop_duplicates(subset=['fecha'], keep='first')
index_acc = df.index[df.loc[:, 'hay_accidente'] == True]

p_acc = index_acc[0]
u_acc = index_acc[-1]
df.loc[:, 'accident_next'] = df.loc[index_acc, 'fecha'].shift(-1)

df.loc[u_acc, ['accident_next']] = (df.loc[u_acc, 'accident_next']).ffill().infer_objects(copy=False)
df.loc[u_acc, ['accident_next']] = df.loc[df.index[-1], 'fecha'] + pd.Timedelta(hours = 2)
df.loc[p_acc, ['accident_next']] = df.loc[p_acc, 'fecha']

df_sub = df.loc[index_acc, to_next].shift(-1).ffill().infer_objects(copy=False)
df[next_c] = np.nan

df_sub.columns = next_c
df.loc[df_sub.index, next_c] = df_sub

df.loc[u_acc, next_c] = df.loc[u_acc, next_c].ffill().infer_objects(copy=False)
df.loc[u_acc, next_c] = [1, 1]
first_v = df.loc[p_acc, to_next]

for i, v in enumerate(first_v):
    df.loc[p_acc, next_c[i]] = v
# raise ValueError(df.loc[:, next_c + ['accident_next']])
df['accident_next'] = pd.to_datetime(df['accident_next'])
df['fecha'] = pd.to_datetime(df['fecha'])

df['accident_next'] = (df['accident_next'] - df['fecha']).dt.total_seconds() / 60
df.loc[index_acc, 'accident_next'] = 0
columns_to_fill = next_c + ['accident_next']

df[columns_to_fill] = df[columns_to_fill].fillna(0)
```

- Se codifican y escalan los valores de manera que sea interpretado de manera correcta por el modelo matemático.
 - Se decide cómo escalar y codificar varias de las variables en función de su características o significado intrínseco. Por ejemplo para variables con distribución standard a priori, se establece un escalado standard.
 - Para valores acotados con distribución continua se escala según un mínimo y un máximo ya que las redes funcionan mejor con valores acotados cerca del 0.
 - Para columnas que finalmente no se ajustan al modelo se pueden eliminar.
 - Las variables no acotadas y que en el futuro se pueden prolongar (como el año de la medición) se le da un nuevo 0.

```
COLS_G_STD = ['intensidad', 'ocupacion', 'carga', 'vmed',
              'temperature_2m', 'relative_humidity_2m', 'dew_point_2m',
              'precipitation', 'rain']
COLS_G_MAX = ['sunshine_duration', 'month', 'day', 'hour', 'cluster']
COLS_G_INT = ['cluster', 'accident_next', 'accidentados_next', 'accidentes_next', 'gravedad_next', 'is_day',
              'hay_accidente']
COLS_DROP = ['vmed']

COLS_CYCLIC = ['month', 'day', 'hour', 'Dia_semana']

dic = 'medidas'
year0 = 2023
lim_accidentados = 5
Sscaler = StandardScaler()
Mscaler = MinMaxScaler()
```

- También para valores cíclicos (temporales) se establecen unas variables artificiales que ayudan a poder relacionar la naturaleza cíclica de la magnitud.

```
for nodo, dict_ in G.nodes.items():
    for col in dict_[dic].columns:
        if col in COLS_G_STD:
            G.nodes[nodo][dic][col] = Sscaler.fit_transform(dict_[dic][col])
        elif col in COLS_G_MAX:
            G.nodes[nodo][dic][col] = Mscaler.fit_transform(dict_[dic][col])
        elif col in COLS_G_INT:
            G.nodes[nodo][dic][col] = dict_[dic][col].astype(int)
        if col == 'year':
            G.nodes[nodo][dic][col] = G.nodes[nodo][dic][col] - year0
        elif col in COLS_DROP:
            G.nodes[nodo][dic].drop(columns=col)
        if col in COLS_CYCLIC:
            G.nodes[nodo][dic]=cyclic_encoder(G.nodes[nodo][dic],col)
    dict_[dic].loc[dict_[dic]['accidentados_next']>lim_accidentados,'accidentados_next'] = lim_accidentados
```

- Por último, debido a un probable desbalance en las escalas de la variable de salida, la variable de tiempo hasta el siguiente accidente se decide pasar a un formato de clasificación, en el que se divide el riesgo de accidente de categoría 0 (accidente), 1 (inminente), 2 (cercano) a 3 (no hay riesgo previsible). Esto requiere de un balance de las clases de salida, en función del resultado de este proceso. En función de los accidentes en el nodo dado, se eligen el mismo número de las otras categorías.

```
categorias_acc = ['accidente', 'inminente', 'cercano', 'no_previsible']
num_acc = [0, 1, 2, 3]
bins_acc = [-1, 15, 45, 120, 1e12]
```

```
binned = pd.cut(dict_[dic]['accident_next'], bins = bins_acc, labels = num_acc)
dict_[dic].loc[:, 'accident_next'] = binned.astype(np.int32)

dict_[dic] = balance_tiempos_cat(dict_[dic], num_acc)
```

```
for nb in num[1:-1]:# quitamos el 0 y el 3 que ya están considerados
    # Filtrar los índices correspondientes a la categoría actual

    index_cat = df.loc[df['accident_next'] == nb].index
    # Seleccionar aleatoriamente `num_ceros` índices, o menos si no hay suficientes
    num_sel = min(len(index_cat), N0)
    filtro_cat = np.random.choice(index_cat, num_sel, replace = False)

    # Asignar 1s en la máscara en las posiciones seleccionadas

    df.loc[filtro_cat,'pad'] = True

return df
```

- Comprobaciones temporales y de coherencia.
 - Se comprueba en varias fases del proceso la correcta dimensión del grafo para todos los nodos.

```
try:
    series_dict = {n: data['medidas'].fecha for n,data in G.nodes.items()}
except Exception as e :
    print('fechas es el indice')
    print(e)
    series_dict = {n: data['medidas'].index for n, data in G.nodes.items()}

tamanos = [index.size for n, index in series_dict.items()]

if len(set(tamanos)) == 1:
    print("Todas las series tienen el mismo número de fechas.")
```

Tras este proceso se llega a un Dataset que se guarda para su posterior uso en el modelo.

```
print('FINAL CHECK')
equal_size_graph(G)

with open(DS_file,'wb') as file:
    pickle.dump(obj=[G, acc_cluster],file)

print(f'DATASET GUARDADO en {os.getcwd()}')
print('___++_')
return G, acc_cluster,locals()
```

4.3.1.3. Desarrollo del modelo

Tras toda la documentación y decisiones tomadas posteriores al estudio del marco teórico del apartado 3, se implementa el modelo de predicción Espaciotemporal en base al grafo obtenido previamente.

Para ello se implementará con lenguaje de programación Python. Usando las librerías adaptadas para redes neuronales y tensores multi espaciales PyTorch y su librería orientada al tratamiento de grafos Pytorch Geometric. Esta librería optimiza los cálculos aprovechando el hardware del dispositivo que usemos de manera que aprovecha lo máximo posible el uso de gpu para los tensores, lo cual es lo óptimo. Se han seguido los pasos que se detallan a continuación.

- Definición de la arquitectura del modelo.
 - El modelo descrito presenta una estructura mixta una parte de convolución espacial pura y otra parte a continuación de celdas LSTM en las cuales entran únicamente los datos relativos a los accidentes y los datos relevantes a los mismos.
 - Se trata de un modelo multisalida en el que a través de los datos descritos anteriormente, se intentará predecir:
 - Cuándo ocurrirá el próximo accidente
 - Número de personas accidentadas
 - Gravedad media
 - Cabe recalcar que en ninguna fase del modelo se pierde la estructura de grafo. Por lo tanto, todas las predicciones serán relativas al nodo en cuestión.
 - La estructura es como sigue:
 - Fase Gráfica
 - 2 capas GCN apiladas en T grafos siendo T el número de pasos temporales. Después son desafiladas, habitando así un bucle por nodo.
 - Tras cada GCN Una capa de no linealidad Relu
 - 1 capa Dropout para evitar Overfitting y mejorar la generalización del modelo
 - Fase LSTM

Se basa en el filtrado por nodo de los pasos con accidente mediante las máscaras. Esto requiere un tratamiento especial ya que cada nodo posee un número diferente de accidentes.

 - 2 capa LSTM (en cada nodo)

- 1 capa de no linealidad Relu
- 1 capa de Dropout para hacer lo propio en cada nodo
- 1 capa Fully Connected
- Fase Padding para uniformizar las dimensiones y concatenación de salidas.

Tras la definición del modelo, se han de adaptar los datos a la entrada necesaria:

- Paso de Grafo a Tensor y extracción de columnas auxiliares para el balanceo de clases: Creación de máscaras para el paso por la celda temporal LSTM.

```
T = len(G.nodes[0]['medidas']) # Número de instantes de tiempo
N = len(G.nodes) # Número de nodos
Y = len(LABEL_COLS) # Número de etiquetas por nodo

features_np = np.zeros(shape=(T, N, len(FEATURE_COLS)), dtype=np.float32)
Ys_np = np.zeros(shape=(T, N, Y), dtype=np.float32)
masks_np = np.zeros(shape=(T, N), dtype=bool)

indexes = []
for i, (node, data_) in enumerate(G.nodes.items()): # 'i' es el índice secuencial del nodo
    medidas = data_['medidas']
    fechas = medidas['fecha']
    indexes.append(fechas)

    Y_vals = medidas[LABEL_COLS].values
    mask_vals = medidas['pad'].values

    node_features = medidas[FEATURE_COLS].values

    features_np[:, i, :] = node_features
    Ys_np[:, i, :] = Y_vals
    masks_np[:, i] = mask_vals

features_t = torch.tensor(features_np)
Ys_t = torch.tensor(Ys_np)
masks_t = torch.tensor(masks_np)

edge_index = torch.tensor(list(G.edges())).t().contiguous()

return features_t, Ys_t, masks_t, edge_index, indexes
```

- Subdivisión en conjuntos de entrenamiento, prueba y validación.


```
def split_data_temporal(features, Ys, masks, indexes, days): 2 usages
    # Asumimos que todos los nodos tienen las mismas fechas
    fechas = indexes[0]

    corte1 = fechas.iloc[0] + pd.Timedelta(days = days[0])
    corte2 = corte1 + pd.Timedelta(days = days[1])

    mascara_train = fechas < corte1
    mascara_val = (fechas >= corte1) & (fechas < corte2)
    mascara_test = fechas >= corte2

    X_train = features[mascara_train]
    X_val = features[mascara_val]
    X_test = features[mascara_test]

    y_train = Ys[mascara_train]
    y_val = Ys[mascara_val]
    y_test = Ys[mascara_test]

    mask_train = masks[mascara_train]
    mask_val = masks[mascara_val]
    mask_test = masks[mascara_test]

    return X_train, X_val, X_test, y_train, y_val, y_test, mask_train, mask_val, mask_test
```

- Implementación de técnicas para la optimización de recursos y tiempo de procesamiento.
 - Librerías CUDA

CUDA es una plataforma de computación paralela y una API (interfaz de programación de aplicaciones) desarrollada por NVIDIA que permite a los desarrolladores utilizar la GPU (Unidad de Procesamiento Gráfico) para realizar cálculos generales de propósito (Nvidia, 2024). Aprovechando la disponibilidad de un equipo con GPU Nvidia dedicada, es posible hacer uso de sus virtudes:

- Programación paralela
- Bibliotecas de alto rendimiento
- Compatibilidad con Pytorch
- Depuración y optimización

- Batch Normalization

Debido a la complejidad del modelo, y al uso intensivo de memoria el uso de Batch no es posible, ya que aumenta el uso de recursos, pero está previsto en algunos de los pasos del modelo.

- Limpieza de Caché y variables y restos de procesamiento.

Debido a limitaciones de Hardware, es necesario limpiar de elementos innecesarios y residuales los espacios de memoria. (Pytorch, 2024)

- Implementación de Precisión Mixta .

GradScaler es una clase proporcionada por PyTorch para manejar la precisión mixta durante el entrenamiento de modelos de aprendizaje profundo. La precisión mixta combina el uso de tipos de datos de precisión simple (32 bits) y de media precisión (16 bits) para optimizar el uso de memoria y mejorar el rendimiento en GPU. (Pytorch, 2023)

- Implementación de reducción de overfitting

- Reducción dimensional

Aún no teniendo demasiada influencia, ya que lo que marcará más serán las dimensiones internas de la red y esto será establecido por otros parámetros como la optimización o las propias limitaciones Hardware, Se establece un proceso PCA para reducir la dimensionalidad del modelo.

- Optimización de hiper parámetros.

Se implementa un proceso Optuna en 2 etapas en las cuales primero se hace una barrida amplia con un porcentaje de los datos para centrar la búsqueda.

- Regularización y dropout

Se implementan capas Dropout en el Modelo para mejorar el aprendizaje y la generalización del modelo

- Early Stopping

El proceso se parará de forma sencilla en caso de que haya indicios de sobreajuste tras unas épocas dadas, de esta manera evitamos que el modelo entre en pérdida.

- Programadores de la tasa de aprendizaje

Para evitar la caída en Valles dentro del espacio latente de la pérdida. Se declara un un programador de cambio en la tasa de aprendizaje de manera que si entra en una zona de decaimiento de la pérdida en validación, se activa un cambio de tasa de aprendizaje y así ayudar al modelo a salir del valle.

- Proceso de optimización de hiper parámetros mediante la librería Optuna.

La librería Optuna es una librería diseñada para el optimizado de parámetros de forma automática y con una API sencilla que permite minimizar dentro de lo posible el proceso de búsqueda de hiperparámetros

- Como hemos detallado anteriormente se implementa, con un doble propósito de mejora del modelo en cuanto a precisión y en cuanto a sobreajuste, una búsqueda con Optuna en 2 fases una fase que barra en un entorno general y una segunda que barra de una forma más exhaustiva en los entornos óptimos de la primera
- Se intentan optimizar parámetros internos del models minimizando la pérdida total:
 - Tasa de aprendizaje
 - Características ocultas
 - Niveles de Dropout
 - Tasa de decaimiento en la regularización
- Creación de secuencias para el paso por la celda temporal: Cada elemento que entra a la parte LSTM genera una secuencia con los pases anteriores al mismo, de esta manera el modelo LSTM puede generalizar mejor obteniendo más contexto.

```
def generate_sequences_node(x, masks, sequence_length = 5): 1 usage

    x = torch.as_tensor(x)
    masks = torch.as_tensor(masks, dtype = torch.bool)

    num_rows, num_cols = x.shape
    device = x.device

    if masks.shape[0] != num_rows:
        raise ValueError(
            f"El número de filas en masks ({masks.shape[0]}) debe coincidir con el número de filas en x ({num_rows})")

    indices = torch.arange(num_rows, device = device).unsqueeze(1).expand(-1, sequence_length)
    offsets = torch.arange(sequence_length - 1, -1, -1, device = device).unsqueeze(0).expand(num_rows, -1)
    sequence_indices = indices - offsets

    mask_expanded = masks.expand(-1, sequence_length)
    masked_indices = torch.where(mask_expanded, sequence_indices, 0)
    valid_indices = torch.clamp(masked_indices, min = 0, max = num_rows - 1)

    sequences = x[valid_indices]

    final_mask = (masked_indices >= 0) & (masked_indices < num_rows) & mask_expanded
    sequences = torch.where(final_mask.unsqueeze(-1), sequences, torch.zeros_like(sequences))

    return sequences[masks.any(dim = 1).squeeze()]
```

- Variables de salida, entrenamiento y resultados.

Se evalúa el modelo con el conjunto de validación y se prueba con el conjunto Test. A partir de los dataset correspondientes se puede graficar la evolución del modelo en sus épocas, nodos y salidas.

Para tratar con este lenguaje es necesario transformar el Grafo obtenido previamente hacia un tensor de unas dimensiones regulares, debido a este requisito ya se había adaptado, la forma del grafo debidamente.

En cuanto a las celdas temporales, es LSTM Sabemos que. Funcionan con estructuras secuenciales. Para ello se establece una ventana de tiempo en la que cada accidente tiene una secuencia anterior de 5 medidas a partir de las cual es el El modelo aprenderá.

También es necesario dividir el tensor en subconjuntos de Entrenamiento validación y test. Esta división no es trivial y se diferencia de las divisiones aleatorias que se hacen normalmente en las redes neuronales. Ya que el orden en variable temporal puede afectar en el modelo y por tanto se establece una división secuencial del Dataset.

Por último antes del entrenamiento también es necesario la creación de Batches de entrenamiento para una mejor agilidad del modelo.

4.3.2. Evaluación y resultados

Tras aplicar todo lo anteriormente expuesto al grafo y obtener los subconjuntos de Entrenamiento Validación y Prueba y se aplica el modelo descrito en este documento para la predicción de tiempo restante, número de personas accidentadas, y gravedad relativas al siguiente accidente en las carreteras de la ciudad de Madrid se obtienen los resultados que se indican a continuación teniendo en cuenta unos hiperparámetros principales dados de :

Dataset

- Número de Nodos/Clusters = 200
- Tipo de grafo = Árbol de Mínima expansión (MST)
- Porcentaje Entrenamiento/Validación/Prueba = 70/20/10
- Límite de accidentados: +5 (5 o más es considerado de la misma manera)
- Categorías de cercanía a accidente en la variable de salida = 4
 - 0: Accidente
 - 1: Inminente 15 - 45 minutos
 - 2: Cercano 45-120 minutos
 - 3: No previsible +120 minutos

Modelo

- Dimensiones PCA: 18

- Optimización de Hiperparámetros en 2 fases
 - Trials F1/F2: 10/20
 - Epochs F1/F2: 15/30
 - Proporción de los datos F1/F2: 30%/70%
 - Tasa de Aprendizaje inicial: 0.0005 - 0.01
 - Decaimiento L2 $1e-5$ - 0.01
 - Capas ocultas 64 (128 no es posible por restricción computacional)
 - Dropouts 0.25-0.45
- Early Stopping
 - Paciencia: 6
 - Delta: 0.2
- Variación Tasa en valle
 - Factor: 0.3
 - Paciencia: 1/3 de Paciencia de Early Stopping
- Ventana de Secuencia LSTM: 4
- Epochs: 50
- Clases de Salida: 3 salidas categóricas con una función de *Cross Entropy Loss*
 - Cercanía al siguiente accidente
 - Número de accidentados en el siguiente accidente
 - Gravedad del siguiente accidente
- Función de pérdida: Pérdida compuesta ponderada.
 - Pesos en la pérdida
 - Cercanía 80%
 - Accidentados 5%
 - Gravedad 15%

Es muy a destacar que el modelo ofrece un valor para cada una de las 3 salidas en cada nodo, es decir que para cada entrada necesitaremos una entrada completa de los 200 nodos, así como que se nos ofrece cada salida en cada nodo de manera que podemos estudiar la cercanía/ nº de accidentados/ gravedad del siguiente accidente en cada nodo separadamente .

4.3.3. Limitaciones, posibles mejoras y consideraciones

Este modelo desde su concepción ha sido expuesto a múltiples limitaciones formales para obtener un sistema válido para una red neuronal. Con lo cual se ve expuesto a múltiples limitaciones que pueden ser objeto de mejora:

1. Relativos a los propios datos
 - a. Desde su concepción la recogida de datos correcta presenta múltiples dificultades para obtener unos datos completos, gratuitos y fiables. Es por esto que el modelo original preveía diferentes y más variadas características, las cuales no se fue capaz de obtener
 - b. Los datos obtenidos presentan por el lado de los sensores demasiados datos y muy populares y redundantes, pero por otro, la variable de salida es escasa , con lo cual el tiempo de procesar los datos se pierde al agregarlos todos después.
 - c. La temporalidad de las mediciones es cada 15 minutos lo cual dada la naturaleza imprevisible de los accidentes quizás sea demasiado para una buena estimación del riesgo de accidente como en otros trabajos hemos visto (1 minuto, 5 minutos...).
 - d. Predicción de eventos y desbalanceo de clases, los accidentes, gravedad y accidentados son datos desbalanceados, no obstante no podemos aumentar datos de salida ya que precisamente una variable clave a tener en cuenta es la frecuencia con la que se dan los eventos
2. Relativos a la transformación de los datos a Grafo
 - a. El agregado de datos en nodos se establece de forma arbitraria (num_nodos = 200) y con un método sencillo el cual puede ser mejorado (Kmeans).
 - b. El hecho de tener accidentes variables en cada nodo del grafo dificulta enormemente el procesado del mismo y el tiempo de procesamiento.
3. Relativos al tipo de modelo
 - a. Las redes neuronales se manejan en forma de tensores regulares lo cual hace que en nuestro caso con accidentes variables en cada nodo fuese difícil de tratar y es necesario una secuenciación, padding etc.
 - b. El modelo inicialmente pensado no es posible de aplicar por esta razón lo cual ha derivado en la creación de un modelo 'custom' el cual no está testeado ni probado su validez y/o precisión.
4. Relativos a la naturaleza de la clasificación
 - a. Debido a que la variable de salida se ha discretizado de un tiempo al accidente hasta un riesgo de accidente, se ha perdido precisión en los datos, no obstante se gana en homogeneidad y facilidad y los datos son mejor balanceados.

5. Referencias

- Chen, K. (2020). Dynamic Spatio-Temporal Graph-Based CNNs for Traffic Flow Prediction. 8. <https://ieeexplore.ieee.org/document/9207934>
- Ciresan, D., M, U., & Schmidhuber, J. (2012). Multi-column deep neural network for traffic sign classification. *Neural NetWorks*, 32. 1.
https://scholar.google.com/citations?view_op=view_citation&hl=es&user=gLnCTgIAAAAJ&cstart=200&pagesize=100&sortby=pubdate&citation_for_view=gLnCTgIAAAAJ:8uzoZH4hB9AC
- Comunidad de Madrid. (2023). *Dossier de tráfico Madrid 2022* [Informe de autoridad de tráfico de la comunidad de Madrid con los datos de tráfico de la comunidad de Madrid durante el año 2022].
<https://www.comunidad.madrid/sites/default/files/aud/transportes/dossier20221.pdf>
- Comunidad de Madrid. (2023, 05 22). *Informe El impacto de la enfermedad traumática grave en España*. Los siniestros de tráfico provocan 4 de cada 10 ingresos por lesión grave en UCI, sobre todo en hombres de 26 a 35 años.
<https://www.comunidad.madrid/noticias/2023/05/22/siniestros-traffic-provocan-4-cada-10-ingresos-lesion-grave-uci-todo-hombres-26-35-anos>
- Crick, F. (1989). The recent excitement about neural networks.
<https://doi.org/10.1038/337129A0>
- Dan, C., Giusti, A., Gambardella, L. M., & Schmidhuber, J. (2013). *Mitosis Detection in Breast Cancer Histology Images with Deep Neural Networks*.
https://link.springer.com/chapter/10.1007/978-3-642-40763-5_51
- Dayhoff, J. E., & DeLeo, J. M. (2020). Artificial Neural Networks.
[https://acsjournals.onlinelibrary.wiley.com/doi/10.1002/1097-0142\(20010415\)91:8+%3C1615::AID-CNCR1175%3E3.0.CO;2-L](https://acsjournals.onlinelibrary.wiley.com/doi/10.1002/1097-0142(20010415)91:8+%3C1615::AID-CNCR1175%3E3.0.CO;2-L)
- Deretić, N. (2022). SARIMA Modelling Approach for Forecasting of Traffic Accidents. *MDPI*. <https://www.mdpi.com/2071-1050/14/8/4403>
- Google Deepmind. (2021). ETA Prediction with Graph Neural Networks in Google Maps. <https://arxiv.org/pdf/2108.11482>
- Jiang, W., & Luo, J. (2022). *Graph Neural Network for Traffic Forecasting: A Survey*. <https://arxiv.org/pdf/2101.11174>

- Jin, M. (2024). A Survey on Graph Neural Networks for Time Series: Forecasting, Classification, Imputation, and Anomaly Detection. <https://www.semanticscholar.org/reader/d3dbbd0f0de51b421a6220bd6480b8d2e99a88e9>
- Jingyuan, W. (2016). Traffic Speed Prediction and Congestion Source Exploration: A Deep Learning Method. *IEEE*. <https://ieeexplore.ieee.org/document/7837874>
- Kim, T., Lee, H., & Yung, H.-J. (2024). <https://arxiv.org/pdf/2405.18602>
- Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. <https://arxiv.org/pdf/1609.02907>
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>
- Kumar, B. (2021, August 31). *Convolutional Neural Networks: A Brief History of their Evolution*. Medium. Retrieved October 27, 2024, from <https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597>
- Lecun, Y. (1998). Gradient-Based Learning Applied to Document Recognition. http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf
- Li, Y., Yu, R., & Shahabi, C. (2018). Diffusion Convolutional Recurrent Neural Network (DCRNN): DATA-DRIVEN TRAFFIC FORECASTING. *ICLR*. <https://arxiv.org/pdf/1707.01926>
- Li-Minn Ang, K., & Kah Phooi Seng, J. (2022). Emerging Technologies for Smart Cities' Transportation: Geo-Information, Data Analytics and Machine Learning Approaches. *International Journal of Geo-Information*. <https://doi.org/10.3390/ijgi11020085>
- Luo, X., Zhu, C., & Zhang, D. (2024). STG4Traffic: A Survey and Benchmark of Spatial- Temporal Graph Neural Networks for Traffic Prediction. <https://arxiv.org/pdf/2307.00495>
- Nvidia. (2024). *CUDA Installation Guide for Windows*. NVIDIA Docs. <https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html>

- Nvidia Blog. (2022). *¿Qué son las Graph Neural Networks?*
<https://la.blogs.nvidia.com/blog/que-son-las-graph-neural-networks/>
- Open AI. (2024). *Chat GPT*. ChatGPT. <https://chatgpt.com/>
- Optuna. (2024). Optuna - A hyperparameter optimization framework. Retrieved 2024, from <https://optuna.org/>
- Pradhan, B., & Sameen, M. I. (2019). *Laser Scanning Systems in Highway and Safety Assessment: Analysis of Highway Geometry and Safety Using LiDAR*. Springer International Publishing.
https://opus.lib.uts.edu.au/bitstream/10453/148686/2/454584_1_En_11_Chapter_Author.pdf
- *Premio Princesa de Asturias de Investigación Científica y Técnica 2022 - Geoffrey Hinton, Yann LeCun, Yoshua Bengio y Demis Hassabis*. (2022). Fundación Princesa de Asturias.
<https://www.fpa.es/es/premios-princesa-de-asturias/premiados/2022-geoffrey-hinton-yann-lecun-yoshua-bengio-y-demis-hassabis.html?texto=trayectoria&especifica=0>
- Pytorch. (2023). *Automatic Mixed Precision examples — PyTorch 2.5 documentation*. PyTorch. Retrieved 2024, from https://pytorch.org/docs/stable/notes/amp_examples.html
- Pytorch. (2024). *torch.cuda.empty_cache — PyTorch 2.5 documentation*. PyTorch.
https://pytorch.org/docs/stable/generated/torch.cuda.empty_cache.html
- *Recurrent Neural Networks (RNN) - The Vanishing Gradient Problem - Blogs*. (2018, August 23). SuperDataScience.
<https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem/>
- Ren, H., Song, Y., & Liu, J. (2017). A Deep Learning Approach to the Prediction of Short-term Traffic Accident Risk.
<https://www.semanticscholar.org/reader/180ed2a4460f7cc12a0ed067f485602e0667ea59>
- Ren, H., Wang, J., & Hu, Y. (2018). A Deep Learning Approach to the Citywide Traffic Accident Risk Prediction.
- Scarcelli, F. (2009). The Graph Neural Network Model.
<https://ro.uow.edu.au/cgi/viewcontent.cgi?article=10501&context=infopapers>

- Schmidhuber, J. (2013). *My First Deep Learning System 1991 / Deep Learning Timeline 1960-2013*. IDSIA.
<https://people.idsia.ch/~juergen/firstdeeplearner.html>
- Schmidhuber, J. (2022). Annotated History of Modern AI and Deep Learning.
<https://arxiv.org/abs/2212.11279>
- Shams Forruque, A. (2024). Enhancement of traffic forecasting through graph neural network-based information fusion techniques - ScienceDirect.
<https://www.sciencedirect.com/science/article/pii/S1566253524002446?via%3Dihub>
- Shi, Y., Biswas, R., & Noori, M. (2021). Predicting Road Accident Risk Using Geospatial Data and Machine Learning.
<https://doi.org/10.1145/3474717.3484253>
- Sirisha, U. M. (2022). Profit Prediction Using ARIMA, SARIMA and LSTM Models in Time Series Forecasting: A Comparison. *IEEE*, 10.
<https://ieeexplore.ieee.org/abstract/document/9964190>
- Stanford University. (2021). *Deep Learning for Graphs*. Machine Learning with Graphs.
<https://www.youtube.com/watch?v=MH4yvtgAR-4&list=PLoROMvodv4rPLKxlpqhjhPgDQy7imNkDn&index=20>
- Strigl, D. (2010). Performance and Scalability of GPU-Based Convolutional Neural Networks | IEEE Conference Publication. *IEEE*.
<https://ieeexplore.ieee.org/abstract/document/5452452>
- Wu, Z., Pan, S., & Chen, F. (2019). *A Comprehensive Survey on Graph Neural Networks*. <https://arxiv.org/pdf/1901.00596>
- Yao, K., Cohn, T., & Vylomova, E. (2015). Depth-Gated LSTM.
@article{Yao2015DepthGatedL, title={Depth-Gated LSTM}, author={Kaisheng Yao and Trevor Cohn and Ekaterina Vylomova and Kevin Duh and Chris Dyer}, journal={ArXiv}, year={2015}, volume={abs/1508.03790}, url={https://api.semanticscholar.org/Corpus
- Ying, R., He, R., & Chen, C. (2018). Graph Convolutional Neural Networks for Web-Scale Recommender Systems. <https://arxiv.org/pdf/1806.01973>
- Yu, L. (2021). Deep spatio-temporal graph convolutional network for traffic accident prediction. <https://sci-hub.se/10.1016/j.neucom.2020.09.043>

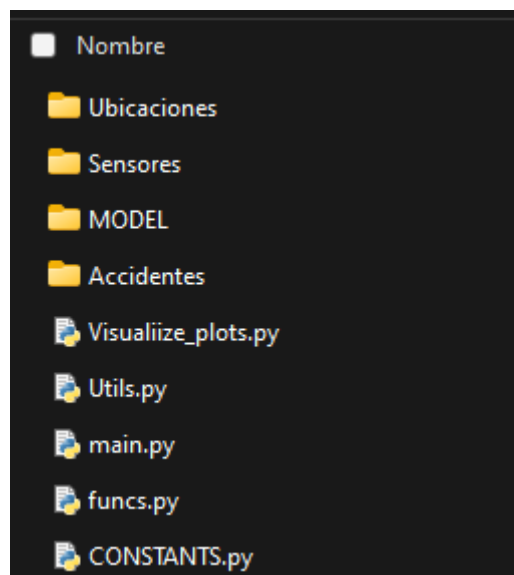
- Yuan, F. (2023, May 29). *Graph Neural Networks (GNNs): Comparison between CNNs and GNNs*. Medium. Retrieved October 27, 2024, from <https://medium.com/@ds-faxi-yuan/graph-neural-networks-gnn-comparison-between-cnn-and-gnn-5c97fdb3e31>
- Zhang, W., Yinghao, Y., & Yong, K. (2019). Short-term traffic flow prediction based on spatio-temporal analysis and CNN deep learning. *Transportmetrica A: Transport Science*.
<https://www.tandfonline.com/doi/full/10.1080/23249935.2019.1637966>
- Zhao, L., Son, Y., & Zan, C. (2015). T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. <https://arxiv.org/pdf/1811.05320>
- Zhou,, J., & Cui, G. (2018). Graph Neural Networks: A Review of Methods and Applications. <https://api.semanticscholar.org/CorpusID:56517517>

Apéndice I : Código Dataset de Grafo Espacio temporal

A continuación se expone una breve descripción de la estructura del código relacionada con el Dataset, las funciones y se desglosa el código completo.

Estructura de Ficheros

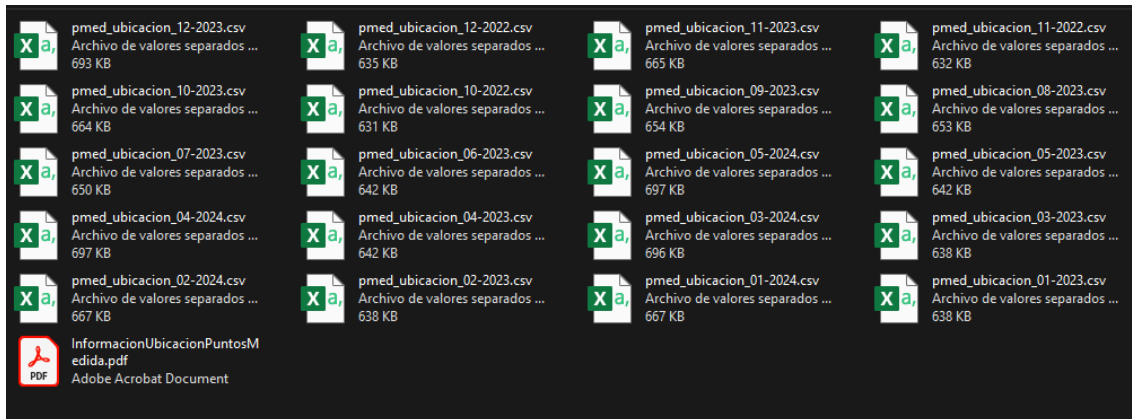
La estructura es la que sigue:



Donde las carpetas Ubicaciones, Sensores y Accidentes son las carpetas con los CSV de origen de los datos de las Ubicaciones de los sensores de tráfico, sus mediciones y los datos de los accidentes, respectivamente. Además posee archivo CSV del calendario laboral.













Carpeta Ubicaciones

Contiene los CSV que serán leídos para las posiciones geospaciales de los sensores.
Ordenados por meses.



Carpeta Sensores

Contiene los archivos (bastante masivos) de las mediciones de tráfico, así como una subcarpeta con los datos propiamente dichos para ser leídos por el código correspondiente.

	Sensores	22/09/2024 15:57	Carpeta de archivos	
	.cache.sqlite	03/09/2024 0:36	Archivo SQLITE	100.424 KB
	01-2023.zip	21/09/2024 15:22	Carpeta comprimi...	78.687 KB
	02-2023.zip	21/09/2024 15:21	Carpeta comprimi...	71.818 KB
	03-2023.zip	21/09/2024 15:21	Carpeta comprimi...	79.641 KB
	04-2023.zip	21/09/2024 15:21	Carpeta comprimi...	76.938 KB
	05-2023.zip	14/08/2024 0:06	Carpeta comprimi...	80.912 KB
	10-2022.zip	21/09/2024 15:32	Carpeta comprimi...	77.074 KB
	11-2022.zip	21/09/2024 15:32	Carpeta comprimi...	74.801 KB
	12-2022.zip	21/09/2024 15:32	Carpeta comprimi...	77.489 KB
	12-2023.zip	22/09/2024 15:57	Carpeta comprimi...	93.974 KB
	Estructura_DS_Contenido_Trafico_Hist...	21/09/2024 18:08	Adobe Acrobat D...	969 KB

Nombre	Fecha de modificación	Tipo	Tamaño
01-2023.csv	21/09/2024 20:03	Archivo de valores...	735.262 KB
01-2024.csv	12/02/2024 15:59	Archivo de valores...	776.117 KB
02-2023.csv	21/09/2024 20:03	Archivo de valores...	662.054 KB
02-2024.csv	04/03/2024 12:27	Archivo de valores...	729.523 KB
03-2023.csv	21/09/2024 20:03	Archivo de valores...	741.207 KB
03-2024.csv	01/04/2024 18:09	Archivo de valores...	782.017 KB
04-2023.csv	21/09/2024 20:03	Archivo de valores...	723.090 KB
04-2024.csv	10/06/2024 15:15	Archivo de valores...	757.885 KB
05-2023.csv	09/06/2023 19:40	Archivo de valores...	752.494 KB
05-2024.csv	10/06/2024 17:44	Archivo de valores...	777.537 KB
06-2023.csv	03/07/2023 12:54	Archivo de valores...	727.373 KB
07-2023.csv	28/08/2023 15:04	Archivo de valores...	752.449 KB
08-2023.csv	04/09/2023 16:12	Archivo de valores...	737.307 KB
09-2023.csv	04/10/2023 15:23	Archivo de valores...	728.573 KB
10-2022.csv	21/09/2024 20:03	Archivo de valores...	728.106 KB
10-2023.csv	02/11/2023 17:28	Archivo de valores...	753.905 KB
11-2022.csv	21/09/2024 20:03	Archivo de valores...	704.040 KB
11-2023.csv	18/12/2023 15:14	Archivo de valores...	743.653 KB
12-2022.csv	21/09/2024 20:03	Archivo de valores separados por comas de Microsoft Excel	
12-2023.csv	17/09/2024 10:18	Archivo de valores...	769.271 KB

Mientras que los archivos de ejecución en PYthon son:

main.py

Se encarga del procesado principal durante todo el flujo de ETL

funcs.py

Contiene las funciones necesarias para el desarrollo del proceso

Utils.py

Una serie de funciones auxiliares generales para uso dentro del proceso

CONSTANTS.py

En este archivo se guardarán todas las constantes necesarias para el discurso del modelo para tenerlas parametrizadas debidamente.

Todo el código original se desplegará en un repositorio WEB en GITHUB con la URL de acceso público : [JoseMaNi/TFM_STGCN_LSTM_Node: Dataset y modelado de una red convolutiva de grafos con LSTM por nodo](https://github.com/JoseMaNi/TFM_STGCN_LSTM_Node_Dataset_y_modelado_de_una_red_convolutiva_de_grafos_con_LSTM_por_nodo)

Estructura del código principal main.py

El código que presenta el archivo main sigue la siguiente estructura:

El código llama a la función “medidas_sensores”, que extrae y procesa los datos de las ID los sensores de tráfico, asignando las medidas a los nodos correspondientes del grafo espacio-temporal.

Se tienen 2 fuentes de datos:

Datos de ubicación de los sensores (DS_folder_ubi): contienen la ubicación geoespacial de los sensores, para asociarlos a nodos específicos en el grafo.

Datos de las mediciones de los sensores (DS_folder_sens): las mediciones reales registradas por los sensores.

En este contexto, la variable “medidas” contiene las mediciones procesadas que luego se integrarán en el grafo, y “df” contiene información geoespacial sobre los sensores, que será útil para la asignación de nodos.

- Función “medidas_sensores”

Esta función consta de Varias partes

1. Descompresión de archivos de datos: El script comienza descomprimiendo los archivos de datos (función unzip_csv) donde están almacenadas las mediciones.
2. Carga y limpieza de datos de ubicación (DS_folder_ubi): Se leen archivos que contienen información sobre la ubicación de los sensores (archivos que empiezan con "pmed_").
3. Formateo: La función utiliza las columnas definidas en “cols_v” y ajusta el formato de latitud y longitud ya que están en formato de cadena de texto. Esta transformación asegura que las coordenadas geográficas sean válidas y utilizables para asignar los sensores a nodos del grafo.
4. Concatenación y eliminación de duplicados Se concatenan los datos de todos los archivos leídos en un solo DataFrame (sens_). Luego, se eliminan los duplicados en base al identificador del sensor (id), creando un DataFrame único de ubicaciones de sensores (sens_unico).
5. Extracción de medidas: La función invoca a otra función importante: “extraer_medidas”, que procesa las mediciones reales registradas por los

sensores y las carga. Después se intersecan las IDs con las previamente obtenidas. Esto se hace para asegurarnos que se seleccionan sensores que existen. En este apartado se seleccionan los sensores cuyas mediciones están disponibles en los archivos de mediciones y se intersecan con los sensores únicos de “sens_unico” (usando los IDs de las ubicaciones y los sensores, para asegurar el proceso).

6. Filtrado de datos: Se eliminan filas con valores nulos, dejando solo los datos completos y limpios listos para ser usados en el grafo.

- Clustering

- Tras lo anterior se llama a la función “Clustering”, que realiza el proceso homónimo en función de la variable num_clusters mediante el método de Kmeans.
- En esta misma función a raíz de los puntos espaciales hallados en los clusters se obtienen las distancias entre ellos.

Usando las distancias entre nodos, se emplea el método del árbol de mínima expansión (MST) y así obtener la matriz de adyacencia del grafo de una manera ligera. y así armar el grafo. tras esta función obtenemos:

- Centers: Las coordenadas geográficas de los centros de cada cluster, que representan posiciones clave en el grafo.
- G: Un grafo generado mediante un árbol de expansión mínima , que conecta los clusters de forma eficiente.

- Limpieza

En la función “limpieza_medidas” se realiza un proceso de limpieza de las medidas defectuosas que puedan dar error en futuros procesos como la agregación que haremos para sacar valores medios de cada cluster en cada paso temporal en cada nodo.

- Filtro de fechas

Tras esto se intenta filtrar esos datos sólo donde haya medidas con accidente para intentar reducir los datos al mínimo relevante.

Para saber a qué nodo corresponde cada accidente, se utiliza un método de cercanía geoespacial para determinar el nodo más cercano, ya tenemos localizados los accidentes en sus nodos.

Para ello se extraen las fechas donde hubo un accidente (aplicando unos pasos atrás en el tiempo para tener una referencia temporal en forma de ventana deslizante) y se filtran dichas las fechas en todos los nodos.

Para balancear la clase final más importante (el tiempo al próximo accidente) ya que accidentes hay pocos frente a no accidentes, se decide además de escoger y filtrar (filtro_total) únicamente fechas con accidente y el mismo número de las fechas sin accidente, esto será clave para que los accidentes no se vean infrarepresentados.

- Carga de Datos Meteorológicos y Fusión

Tras esto en la función se cargan mediante la API los datos meteorológicos de los centros de los clusters. Se fusionan por clave de fecha con el Grafo principal en clave horaria.

- Transformación del Calendario Laboral y Fusión

También se transforman los datos del calendario laboral mediante una transformación sencilla. También se fusiona con el grafo principal en clave diaria.

- Establecimiento de la Variable de Salida

Para establecer la variable de salida se establece a partir de únicamente los accidentes y una dislocación temporal de una posición (shift) de la hora del accidente siguiente. Tras fusionar con el grafo G los nulls que están en el medio de los puntos con accidentes se rellenan con la última fecha válida posterior, de esa manera tenemos la fecha de siguiente accidente, tras esto restando la hora de cada referencia tendremos el tiempo restante para cada accidente en cada nodo, lo cual es nuestra variable de salida y variable objetivo.

- Preparación de los Datos para la Red Neuronal

Tras esto sólo quedará, transformar las columnas debidamente para su análisis en una red neuronal, normalizando, escalando y discretizando columnas para un mejor resultado con la función "scale_encode".

Apéndice II : Código modelo predicción con Grafos

A continuación se expone una breve descripción de la estructura del código relacionada con el desarrollo del modelo, las funciones y lo necesario para el modelo.

Para encontrar los archivos relativos al modelo de aprendizaje profundo basado en grafos debemos entrar en la carpeta MODEL mostrada en Apéndice I. Dentro de esta carpeta se encuentran:

Nombre	Fecha de modificación	Tipo	Tamaño
.ipynb_checkpoints	03/10/2024 1:26	Carpeta de archivos	
__pycache__	30/10/2024 23:22	Carpeta de archivos	
Nueva carpeta	24/09/2024 2:38	Carpeta de archivos	
__init__.py	03/10/2024 1:23	Python File	0 KB
Ejemplo.py	30/10/2024 1:12	Python File	6 KB
MODEL.py	30/10/2024 0:59	Python File	5 KB
model_funcs.py	30/10/2024 4:08	Python File	17 KB
Optimisation_and_plotting.py	30/10/2024 7:44	Python File	12 KB
preprocessing_G.py	23/10/2024 0:50	Python File	6 KB
reconstruct.py	22/10/2024 0:00	Python File	3 KB

Anexos I : Documentación Fuentes de datos

A continuación se disponen los documentos completos de la fuente pública original de los datos en forma de leyenda.