ANÁLISIS DE MULTAS DE CIRCULACIÓN EN MADRID

CEUPE

MÁSTER DATA SCIENCE

TRABAJO DE FIN DE MÁSTER

María Victoria Perujo Gil

Marzo 2025

ceupe
European Business School

ANÁLISIS DE MULTAS DE CIRCULACIÓN EN MADRID

CEUPE

MÁSTER DATA SCIENCE

TRABAJO DE FIN DE MÁSTER

AUTORA:
María Victoria Perujo Gil

TUTORES:
Cristina Ruiz
David Cañones

Marzo 2025

D.ª: <u>María Victoria Perujo Gil</u> autoriza a que el presente trabajo se guarde y custodie en los repositorios de CEUPE.

Índice de contenidos

Índice de ilustraciones

Índice de códigos

Glosario

WSL (Windows Subsystem for Linux): Subsistema de Windows para Linux que permite ejecutar distribuciones de Linux, como Ubuntu, directamente sobre Windows sin necesidad de una máquina virtual. Es útil para desarrolladores que desean usar herramientas y entornos propios de Linux en Windows.

Ubuntu: Distribución de Linux basada en Debian, ampliamente utilizada por su facilidad de uso y comunidad activa. Es común en entornos de desarrollo y ciencia de datos.

Python: Lenguaje de programación interpretado, de alto nivel, muy utilizado en ciencia de datos, Machine Learning y automatización por su sintaxis sencilla y la gran cantidad de bibliotecas disponibles.

Entorno 'venv': Abreviatura de “virtual environment”. Es una herramienta de Python para crear entornos aislados, permitiendo instalar dependencias específicas para un proyecto sin afectar otras instalaciones de Python en el sistema.

Datos tabulares: Conjunto de datos organizados en filas y columnas, similar a una hoja de cálculo. Cada fila representa una observación y cada columna una variable.

Archivo CSV (Comma-Separated Values): Formato de archivo de texto plano donde los valores están separados por comas. Es ampliamente utilizado para almacenar datos tabulares de forma sencilla y compatible con múltiples programas.

Archivo Excel: Archivo generado por Microsoft Excel (extensión .xls o .xlsx), que permite almacenar datos tabulares, realizar cálculos, y aplicar fórmulas o visualizaciones. Es común en entornos empresariales y académicos.

Microsoft Power BI: Herramienta de análisis y visualización de datos de Microsoft que permite crear dashboards interactivos y realizar informes basados en múltiples fuentes de datos.

Dataset: Conjunto estructurado de datos que se utiliza para análisis o entrenamiento de modelos de Machine Learning. Puede estar en formato tabular, texto, imagen, entre otros.

“Left join”: Tipo de combinación entre tablas en el que se conservan todos los registros de la tabla izquierda y se añaden los registros coincidentes de la tabla derecha. Si no hay coincidencia, se completan con valores nulos.

ML (Machine Learning): Aprendizaje Automático en español. Rama de la inteligencia artificial que permite a los sistemas aprender automáticamente a partir de datos, sin ser explícitamente programados. Utiliza algoritmos que identifican patrones en los datos y pueden hacer predicciones o tomar decisiones basadas en nueva información.

Modelo supervisado: Tipo de algoritmo de ML que aprende a partir de ejemplos etiquetados (con variable objetivo conocida).

Variables: Columnas o atributos de un conjunto de datos. Representan características que describen cada observación (fila).

Variable objetivo: También llamada “target” o “etiqueta”. Es la variable que se quiere predecir mediante un modelo de Machine Learning. Puede ser continua (regresión) o categórica (clasificación).

Features: También conocidas como “variables explicativas” o “predictoras”. Son las variables utilizadas por el modelo de Machine Learning para aprender patrones y realizar predicciones sobre la variable objetivo.

RMSE (Root Mean Squared Error): Raíz del error cuadrático medio. Es una métrica utilizada en problemas de regresión para medir la diferencia promedio entre los valores predichos y los reales. Cuanto menor sea el RMSE, mejor será el rendimiento del modelo.

Embedding: Representación numérica de un texto en un espacio vectorial, de modo que se conserve su significado semántico. Es útil para aplicar técnicas de ML sobre texto.

Regresión multi-output: Tipo de problema en el que se predicen múltiples variables objetivo numéricas simultáneamente.

Overfitting: Fenómeno en el que un modelo aprende demasiado bien los datos de entrenamiento, incluyendo el ruido, y por lo tanto falla al generalizar a datos nuevos.

Underfitting: Cuando un modelo es demasiado simple para capturar la complejidad del conjunto de datos y, por tanto, no aprende correctamente.

Ruido: Datos que no siguen un patrón claro o que no aportan información útil al modelo.

Feature engineering: Ingeniería de características en español. Proceso de creación, transformación o selección de variables que permiten el correcto entrenamiento del modelo.

1. Introducción

El presente Trabajo de Fin de Máster tiene como objetivo principal desarrollar un proyecto analítico “end to end” para la creación de un modelo de Machine Learning capaz de predecir la cantidad de multas de circulación graves y muy graves que se van a producir en un mes concreto, en qué calles y a qué tipología corresponden.

Para ello, resulta fundamental explorar y analizar previamente el dataset y formular las preguntas analíticas adecuadas para comprender mejor la historia del dato y tener claros los pasos a seguir desde el principio. Además, es necesario realizar una serie de transformaciones que faciliten el procesamiento de los datos a la hora de aplicar las técnicas de Machine Learning necesarias.

El proyecto abarca desde la adquisición y limpieza de los datos hasta la creación y validación de un modelo predictivo. Para ello, se han utilizado herramientas de análisis de datos, procesamiento de lenguaje natural (NLP), técnicas de reducción de dimensionalidad, clustering y algoritmos avanzados de Machine Learning como LightGBM, CatBoost, XGBoost y Random Forest. También se ha llevado a cabo una validación posterior del modelo con datos reales no vistos durante el entrenamiento, con el fin de evaluar su rendimiento en escenarios reales.

Todo este proceso permite no solo obtener predicciones precisas, sino también analizar e interpretar las variables del conjunto de datos y sus influencias entre sí. Se han utilizado técnicas de ingeniería de características para generar nuevas variables que capturan patrones temporales y espaciales de las multas.

En definitiva, el presente proyecto no solo tiene como objetivo construir un modelo predictivo, sino también aportar una visión global del ámbito de las multas de circulación en la ciudad de Madrid, entendiendo sus causas, su evolución y sus implicaciones, y demostrando así el valor del análisis de datos aplicado a problemas del mundo real.

2. Objetivos

El objetivo principal de este Trabajo de Fin de Máster es desarrollar un modelo predictivo capaz de calcular la cantidad y el tipo de multas de circulación graves y muy graves que se producirán en cada vía de la ciudad de Madrid en un mes determinado, utilizando técnicas de Machine Learning. Para alcanzar este objetivo general, se establecen los siguientes objetivos secundarios:

- ✓ Comprender en profundidad la información contenida en el dataset.
- ✓ Realizar una limpieza exhaustiva y un modelado adecuado de los datos, aplicando técnicas avanzadas de preprocesamiento, reducción de dimensionalidad y transformación de columnas complejas para preparar los datos para su análisis y posterior creación del modelo del ML.
- ✓ Formular las preguntas analíticas adecuadas que permitan extraer patrones relevantes de comportamiento, estacionalidad y tipología en las multas, con el fin de fundamentar sólidamente la construcción del modelo.
- ✓ Realizar un análisis exploratorio en profundidad de las variables del dataset, para comprender su distribución, relaciones internas, tipologías y utilidad de cara al modelo predictivo, identificando aquellas que aportan mayor valor y descartando las irrelevantes o redundantes.
- ✓ Aplicar técnicas de procesamiento de lenguaje natural (NLP) para convertir el texto libre en variables estructuradas y representativas.
- ✓ Estandarizar, codificar y transformar las variables categóricas y numéricas para construir un dataset optimizado que permita al modelo capturar la complejidad de los targets a predecir.
- ✓ Entrenar, evaluar y comparar múltiples modelos de regresión multi-output con el fin de seleccionar el que ofrezca el menor error en la predicción.
- ✓ Validar el modelo con datos no vistos, simulando un escenario real de predicción futura, para analizar el grado de acierto del modelo y establecer su viabilidad práctica.
- ✓ Extraer conclusiones aplicables al ámbito de la seguridad vial, aportando una herramienta que permita a las autoridades anticiparse a posibles focos de infracciones y optimizar recursos de control y prevención.

3. Metodología

3.1. Origen de los datos

Los datos utilizados en este estudio han sido obtenidos de la página web “Portal de datos abiertos del Ayuntamiento de Madrid. Se trata de una plataforma oficial del Ayuntamiento de Madrid que pone a disposición del ciudadano datos abiertos que puedan ayudarlo en sus proyectos.

Por un lado, se han descargado los datos relacionados con las multas de circulación, que constituye el dataset principal del proyecto. Por otro lado, se ha descargado el callejero, que es un documento de apoyo para la transformación de los datos.

3.2. Descripción del dataset

3.2.1. Multas de circulación (Detalle)

Descripción extraída literalmente del portal:

“En este conjunto de datos se van incorporando todas las multas de circulación que el Ayuntamiento de Madrid tramita cada mes, con todo el detalle posible sobre cada una de ellas que permite la legislación de protección de datos. Se incluye información sobre la infracción cometida, su gravedad o el lugar donde se denunció. La información se ajusta a los datos existentes en el momento de la publicación, y no prejuzgan el resultado final de la tramitación del expediente sancionador, como consecuencia reclamaciones y/ o recursos de las personas denunciadas.”

“Por motivos de protección de datos, existe un número de infracciones que se han agrupado, indicando solamente el número y el importe total de todas ellas. Porcentualmente, es un número ínfimo. Se recuerda, que está expresamente prohibido realizar labores de re-identificación personal de la información.”

“La información que se proporciona es la que se tiene en los sistemas de información del Ayuntamiento, en el momento de la extracción, con lo cual pudieran existir alguna infracción que todavía no se hubiese mecanizado en ese momento, lo cual no quiere indicar que no exista. Este número de infracciones, de existir, seria mínimo.”

“En estos ficheros de detalle, no se contabilizan las denuncias que contienen algún error y tampoco las anuladas.”

“Los ficheros, a partir de marzo 2016 (incluido), presentan dos columnas adicionales, como mejora de la información, con campos que faciliten la georreferenciación (Coordenada-X; Coordenada-Y, en sistema de referencia ETRS89). En una primera fase, estos campos incluyen datos de las multas expedidas por:

- *Rebasar un semáforo en fase roja.*

- *No respetar las señales en una vía de circulación restringida o reservadas.*
- *Acceder a Madrid Central sin autorización.*

En fases sucesivas, se irá ampliando progresivamente la información de coordenadas."

En el portal hay disponibles dos ficheros CSV para cada mes, uno con el detalle y otro con las multas agrupadas/excluidas. Para este proyecto se han utilizado los ficheros con el detalle de los meses desde enero de 2022 a abril de 2024 para entrenamiento, mayo de 2024 para test y junio de 2024 para la validación final.

Los ficheros con el detalle tienen los siguientes campos:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	CALIFICACIO	LUGAR	MES	ANIO	HORA	IMP_BOL	DESCUENTO	PUNTOS	DENUNCIAN	HECHO-BOL	VEL_LIMITE	VEL_CIRCULA	COORDENAC	COORDENADA-Y
2	LEVE	FRANCISCO (	12	2022	09.38	90.00	SI	0	AGENTES DE ESTACIONAR				4414.39	44793.67

Ilustración 1: Campos de los ficheros CSV descargados del portal

- **Columna A**: Calificación. Expresa la calificación de la infracción denunciada en función de su gravedad; leve, grave o muy grave, conforme determina el RDL 6/2015, de 30 de octubre, por el que se aprueba el Texto Refundido de la Ley sobre Tráfico, Circulación de Vehículos y Seguridad Vial (LSV).
- **Columna B**: Lugar. Ubicación del viario público donde ha tenido lugar la infracción denunciada.
- **Columna C**: Mes. Mes del año en que ha sido formulada la denuncia.
- **Columna D**: Anio. Año en que ha sido formulada la denuncia.
- **Columna E**: Hora. Hora y minutos en que ha sido formulada la denuncia.
- **Columna F**: Imp. Bol. Refleja el importe en euros correspondiente a la multa por la infracción cometida en función de su calificación (columna A). Este importe se verá reducido en un 50% si se paga la multa en el plazo de tiempo establecido en la Ley de Seguridad Vial y no se corresponde necesariamente con el que se recaudará efectivamente.
- **Columna G**. Descuento. SI/NO. Indica si el tipo de infracción permite el pago de la multa con reducción del 50% conforme a la LSV para acogerse al procedimiento abreviado.
- **Columna H**: Puntos. Si la infracción cometida implica, conforme a la LSV, la detracción de puntos en el permiso de conducción del infractor, aparecerán los puntos a detraer.
- **Columna I**: Denunciante. Proporciona información sobre la procedencia de la denuncia (Servicio de Estacionamiento Regulado, Agentes de Movilidad, Policía Municipal, Servicio de Apoyo al Control de Estacionamiento, etc).
- **Columna J**: Hecho-Bol. Contiene el texto correspondiente a la infracción denunciada.
- **Columna K**: Vel. Límite. En aquellos casos en que la infracción es un exceso de velocidad, refleja el límite de la vía donde la infracción fue detectada.
- **Columna L**: Vel. Circula. En aquellos casos en que la infracción es un exceso de velocidad, muestra el límite de velocidad medido por el radar.
- **Columna M**: Coordenada X. En aquellos casos en que consta indicado muestra la coordenada del lugar de la denuncia.

- **Columna N**: Coordenada Y. En aquellos casos en que consta indicado muestra la coordenada del lugar de la denuncia.

3.2.2. Callejero (Relación de viales vigentes)

Descripción extraída literalmente del portal:

"Incluye los viales vigentes del término municipal a 31 de diciembre de 2015, ordenada alfabéticamente por denominación de la vía (VIA_NOMBRE)."

Se trata de un fichero CSV que contiene varios campos, pero sólo se necesitaron tres de ellos:

- **VIA_CLASE**: Clase de vía (calle, plaza, etc.).
- **VIA_PAR**: Partícula de la denominación.
- **VIA_NOMBRE**: Denominación de la vía.

Se descargaron todos los archivos del portal para dar inicio a la exploración.

3.3. Entorno y Librerías

El proyecto se llevó a cabo en un ordenador portátil personal (Intel® Core™ i5-9300H CPU @ 2.40GHz, con 16 GB de RAM y Windows 11 Home), por lo que se tuvo que adaptar el procesamiento para que no tuviese demasiado coste computacional.

Para mejorar el rendimiento se usó Visual Studio Code con WSL (Windows Subsystem for Linux) y Ubuntu en lugar de una máquina virtual al uso.

Se creó un entorno 'venv' de Python en el que se instalaron todas las librerías y módulos necesarios para llevar a cabo el proyecto.

A continuación, se detallan las librerías, módulos y herramientas utilizadas, en función de su categoría.

3.3.1. Manipulación y análisis de datos

Pandas: Manipulación y análisis de datos en estructuras tipo DataFrame.

NumPy: Soporte para cálculos numéricos y operaciones con arrays multidimensionales.

Scipy: Operaciones matemáticas avanzadas y optimización.

3.3.2. Visualización de datos

Matplotlib: Creación de gráficos estáticos, animados e interactivos.

Seaborn: Gráficos estadísticos con estilos mejorados sobre Matplotlib.

Plotly: Visualizaciones interactivas de alta calidad.

3.3.3. Machine Learning clásico

Scikit-learn: Biblioteca clave para ML clásico, con algoritmos de clasificación, regresión, clustering y preprocesamiento.

NearestNeighbors (Scikit-learn): Algoritmo basado en la búsqueda de los k vecinos más cercanos en un espacio de datos, útil en clustering, recomendación y detección de anomalías.

OneHotEncoder (Scikit-learn): Codificación de variables categóricas en formato binario.

StandardScaler (Scikit-learn): Normalización de datos mediante estandarización (media 0, desviación estándar 1).

TimeSeriesSplit (Scikit-learn): Validación cruzada es una técnica utilizada para evaluar el rendimiento de un modelo de ML dividiendo los datos en múltiples subconjuntos (folds). En lugar de entrenar y probar el modelo en una única partición de los datos, la validación cruzada divide el dataset en k subconjuntos y entrena el modelo k veces, utilizando diferentes combinaciones de entrenamiento y validación en cada iteración.

Time Series Split de Scikit-learn es un validador cruzado especializado para datos de series temporales. Respeta el orden temporal de los datos, lo que garantiza que siempre entrena con datos pasados y prueba con datos futuros.

Random Forest (Scikit-learn): Algoritmo basado en árboles de decisión para clasificación y regresión. En este proyecto se ha usado regresión multi output.

Este modelo es el resultado de aplicar una técnica conocida como Bagging, que en resumen es una técnica que permite entrenar varios modelos de forma independiente utilizando para el entrenamiento de cada uno de ellos un subconjunto aleatorio de los datos de entrenamiento. Random Forest se puede entender entonces como:

- Se entrenan un número n de árboles independientes entre sí.
- Cada uno de los árboles se entrena con un subconjunto del total de datos de entrenamiento.
- El muestreo será con reemplazamiento, es decir, una misma observación podría acabar en el conjunto de entrenamiento de varios árboles.
- El resultado del modelo será la combinación (la media) de los resultados individuales de cada uno de los árboles.

MultiOutputRegressor (Scikit-learn): Como el problema del proyecto necesita predecir varias salidas, este módulo es útil en los modelos que no admiten de forma

nativa la regresión multiobjetivo. Esta estrategia consiste en ajustar un regresor por cada objetivo.

3.3.4. Modelos de Gradient Boosting

LightGBM: Librería de Gradient Boosting altamente eficiente y escalable que admite muchos algoritmos diferentes, incluidos GBDT, GBRT, GBM y MART. Se demuestra que LightGBM es varias veces más rápido que las implementaciones existentes de Gradient Boosting debido a su método particular de boosting (greedy o codicioso en castellano) y la optimización de la memoria y el cálculo basados en histogramas. Permite aplicar Early Stopping.

XGBoost: XGBoost (eXtreme Gradient Boosting) es una biblioteca de ML distribuida y de código abierto que utiliza árboles de decisión potenciados por gradiente, un algoritmo de boosting del aprendizaje supervisado que hace uso del descenso por gradiente. Es conocido por su velocidad, eficacia y capacidad para escalar bien con grandes conjuntos de datos.

CatBoost: Es una biblioteca de código abierto y alto rendimiento para el refuerzo de gradiente en árboles de decisión. Destaca en el manejo de características categóricas.

3.3.5. Optimización de modelos

Optuna: Librería para la optimización automática de hiperparámetros.

Early Stopping: Técnica para detener el entrenamiento cuando la mejora en la validación se estabiliza, evitando overfitting.

3.3.6. Procesamiento de lenguaje natural (NLP)

spaCy: NLP optimizado para producción con modelos preentrenados.

Transformers (Hugging Face): Modelos de última generación para NLP como BERT y GPT.

SBERT (Sentence-BERT): Modelo basado en BERT para representar frases como embeddings numéricos, útil en NLP avanzado.

Re (Expresiones Regulares): Módulo para limpieza y extracción de patrones de texto mediante expresiones regulares.

RapidFuzz: Coincidencia difusa de texto para encontrar similitudes entre cadenas. Se basa en la concordancia difusa de cadenas, que es una técnica para identificar dos elementos de cadenas de texto que coinciden parcialmente pero no exactamente. Para ello usa el algoritmo de emparejamiento de cadenas, que determina la

proximidad de dos cadenas utilizando la distancia de edición de Levenshtien. Esta distancia calcula lo cerca que están dos cadenas encontrando el número mínimo de "edits" (insertar letra, borrar, cambiar, sustituir) necesarios para transformar una cadena en otra.

3.3.7. Gestión de archivos y codificación

Glob: Módulo de búsqueda de archivos y directorios usando patrones como *.csv.

Unicode: Manejo de caracteres y codificaciones Unicode.

3.3.8. Reducción de dimensionalidad y Clustering

UMAP: Técnica avanzada para reducción de dimensionalidad, similar a t-SNE, pero más rápida.

DBSCAN: Algoritmo de clustering basado en densidad, útil para identificar patrones en datos espaciales o dispersos. No se requiere proporcionar el número de clústeres y puede construir clústeres que tienen una forma arbitraria.

3.4. Análisis inicial y exploratorio de los datos

3.4.1. Creación de muestra del dataset

Debido a las características del ordenador usado para el proyecto, fue imposible trabajar con la totalidad del dataset. Por lo tanto, el primer paso fue generar con pandas y glob una muestra aleatoria del 10% de cada fichero CSV, para tener aproximadamente la misma representación de cada mes en la muestra obtenida. Después, se unieron todas las muestras en un único CSV.

Para ello hubo que limpiar previamente los encabezados de columna con el método .strip(), ya que en algunos ficheros los encabezados tenían espacios vacíos, lo que provocaba duplicidad de columnas.

3.4.2. Análisis inicial

Exploración del dataset

Tras generar el dataset de muestra se utilizó el método .info() para visualizar el resultado:

```
RangeIndex: 530579 entries, 0 to 530578
Data columns (total 14 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   CALIFICACION  530579 non-null  object
 1   LUGAR         530579 non-null  object
 2   MES           530579 non-null  int64
 3   ANIO          530579 non-null  int64
 4   HORA          530579 non-null  float64
 5   IMP_BOL       530579 non-null  float64
 6   DESCUENTO     530579 non-null  object
 7   PUNTOS        530579 non-null  int64
 8   DENUNCIANTE   530579 non-null  object
 9   HECHO-BOL     530579 non-null  object
 10  VEL_LIMITE    530579 non-null  object
 11  VEL_CIRCULA   530579 non-null  object
 12  COORDENADA-X  530579 non-null  object
 13  COORDENADA-Y  530579 non-null  object
dtypes: float64(2), int64(3), object(9)
```

Ilustración 2: Resultado de .info()

Se trata de un DataFrame con 530.579 filas y 14 columnas. Hay 3 tipos de valores: object (9 columnas), int64 (3 columnas) y float64 (2 columnas). No existen valores nulos.

Para conocer cuántos valores únicos hay en cada columna se utilizó el método .nunique():

```
Valores únicos por columna:
CALIFICACION       3
LUGAR          64395
MES               12
ANIO               2
HORA            1440
IMP_BOL           11
DESCUENTO          1
PUNTOS             6
DENUNCIANTE       10
HECHO-BOL        587
VEL_LIMITE        17
VEL_CIRCULA      195
COORDENADA-X    3714
COORDENADA-Y    4037
```

Ilustración 3: Resultado de .nunique()

En el resultado podemos distinguir columnas con alta variedad de valores únicos (LUGAR, HORA, HECHO-BOL, COORDENADA-X, COORDENADA-Y) y columnas con pocos valores únicos (CALIFICACIÓN, MES, ANIO, IMP_BOL, DESCUENTO, PUNTOS, DENUNCIANTE, VEL_LIMITE). En concreto, la columna ‘DESCUENTO’ solo tiene un único valor constante.

Se visualizaron las 20 primeras filas de cada columna con el método .head(20) para tener una idea del contenido. Llamó la atención lo siguiente:

- Los valores están en mayúsculas.
- En la columna 'LUGAR' no existe una estructura unificada para mostrar las clases y nombres de vías. Por ejemplo, las clases de vías aparecen abreviadas en algunos valores, en otros sin abreviar y en otros no aparece. Hay valores con números que corresponden al número de calle o al kilómetro, pero también hay valores sin números. Existe el carácter especial "-" para separar dos vías diferentes (M. CORBERA – RICARDO OTRIZ), lo que sugiere que la vía donde se produce la infracción es la primera y la segunda indica la altura (se produce en M. Corbera a la altura de Ricardo Ortiz).

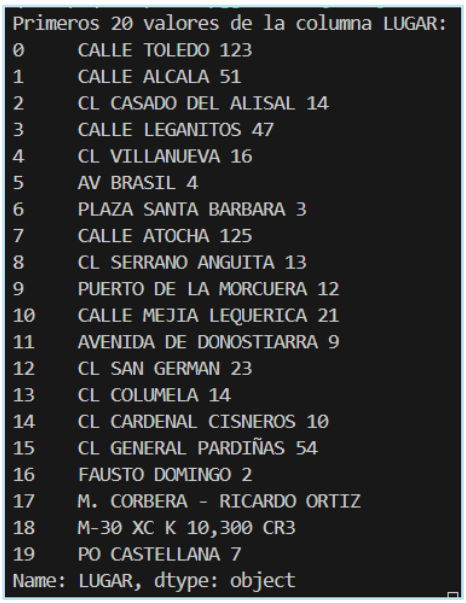


Ilustración 4: Resultado de .head(20) para 'LUGAR'

- La columna 'HORA' tiene formato HH.MM y por eso la detecta como tipo float.
- Los valores de la columna 'IMP_BOL' son realmente enteros y no tipo float.
- La columna 'HECHO-BOL' tampoco tiene una estructura definida, parece un campo "libre" para que el denunciante anote el motivo de la infracción. Existen comas, puntos, acentos, etc.

```
Primeros 20 valores de la columna HECHO-BOL:
0     ACCEDER A LA ZBEDEP DISTRITO CENTRO SIN AUTORI...
1     ACCEDER A LA ZBEDEP DISTRITO CENTRO SIN AUTORI...
2     ESTACIONAR VEHÍCULO DE MOVILIDAD PERSONAL EN L...
3     ACCEDER A LA ZBEDEP DISTRITO CENTRO SIN AUTORI...
4     ESTACIONAR, SIN LA CORRESPONDIENTE AUTORIZACIÓ...
5     ESTACIONAR, SIN LA CORRESPONDIENTE AUTORIZACIÓ...
6     ACCEDER A LA ZBEDEP DISTRITO CENTRO SIN AUTORI...
7     ACCEDER A LA ZBEDEP DISTRITO CENTRO SIN AUTORI...
8     ESTACIONAR CON AUTORIZACIÓN EN LUGAR HABILITAD...
9     ESTACIONAR EN ZONA Y HORARIO DE CARGA Y DESCAR...
10    ACCEDER A LA ZBEDEP DISTRITO CENTRO SIN AUTORI...
11    ESTACIONAR EN DOBLE FILA.                      ...
12    ESTACIONAR CON AUTORIZACIÓN NO VÁLIDA.         ...
13    ESTACIONAR, SIN LA CORRESPONDIENTE AUTORIZACIÓ...
14    ESTACIONAR, SIN LA CORRESPONDIENTE AUTORIZACIÓ...
15    ESTACIONAR, SIN LA CORRESPONDIENTE AUTORIZACIÓ...
16    ESTACIONAR SOBRE LA ACERA.                     ...
17    REBASAR UN SEMÁFORO EN FASE ROJA.              ...
18    SOBREPASAR LA VELOCIDAD MÁXIMA EN VÍAS LIMITAD...
19    ESTACIONAR, SIN LA CORRESPONDIENTE AUTORIZACIÓ...
Name: HECHO-BOL, dtype: object
```

Ilustración 5: Resultado de .head(20) para 'HECHO-BOL'

- En las columnas de velocidad y coordenadas se pueden observar valores en blanco, pero con el .info() ejecutado anteriormente se comprobó que no había valores nulos, por lo que se llega a la conclusión de que, al igual que ocurría en los encabezados, también hay espacios vacíos en los valores del dataset.

Modificaciones básicas

Tras esta exploración inicial se realizaron algunas modificaciones básicas en el dataset:

- Limpieza de espacios vacíos (antes y después del valor) tanto de los encabezados de columna como de los valores de todas las columnas.

```python
# Limpiar espacios en los encabezados y valores de columnas
multas.columns = multas.columns.str.strip()
multas = multas.map(lambda x: x.strip() if isinstance(x, str) else x)
```

Código 1: Limpieza de espacios vacíos con .strip()

- Conversión de encabezados y valores a minúsculas.
- Renombramiento de columnas:
  - 'imp_bol' por 'importe'.
  - 'hecho-bol' por 'hecho'.

```
# Cambiar encabezados a minúsculas y renombrar columnas
multas.columns = multas.columns.str.lower()
multas.rename(
    columns={'imp_bol': 'importe','hecho-bol': 'hecho'}, inplace=True)

# Cambiar todos los valores de los campos a minúsculas
multas = multas.applymap(lambda x: x.lower() if isinstance(x, str) else x)
```

Código 2: Transformación a minúsculas y renombramiento de campos con .lower() y .rename()

- Cambio de tipos de columna:
  - 'importe' de float a int.
  - 'vel_circula' de object a int (cambiando nulos por ceros).
  - 'vel_limite' de object a int (cambiando nulos por ceros).

```
# Cambiar tipos de columna
multas['importe'] = multas['importe'].astype('Int64')
multas['vel_circula'] = pd.to_numeric(
    multas['vel_circula'], errors='coerce').fillna(0).astype('Int64')
multas['vel_limite'] = pd.to_numeric(
    multas['vel_limite'], errors='coerce').fillna(0).astype('Int64')
```

Código 3: Cambio de tipos de datos

- Se consideró interesante para el análisis la creación de una nueva columna 'vel_exceso' con el exceso de velocidad (resta de 'vel_circula' menos 'vel_limite').
- Modificación de la columna 'hora' para que sólo muestre las horas y no los minutos, ya que no interesa hacer un análisis a un nivel tan bajo.

```
# Modificar la columna 'hora' para mostrar sólo las horas
multas['hora'] = multas['hora'].apply(
    lambda x: int(str(x).split('.')[0]) if pd.notnull(x) else x)
```

Código 4: Modificación de la columna 'hora'

- Se decidió prescindir de la columna descuento ya que es un valor constante y, aunque no lo fuese, conocer si una multa podrá pagarse con descuento no aporta nada al modelo concreto que se quería construir, por lo que se eliminó esta columna con el método .drop().

Preguntas analíticas

El siguiente paso fue plantear una serie de preguntas que ayudasen a enfocar la dirección del análisis exploratorio propiamente dicho. Teniendo en cuenta que la variable objetivo sería la cantidad de multas graves y muy graves en un mes, en una vía y de un tipo concretos, cabía preguntarse:

- ¿Cómo se distribuyen las multas según su clasificación?
- ¿En qué meses hay más multas graves/muy graves? ¿Cómo es la curva de la serie temporal?
- ¿Existe una relación directa entre la calificación de la multa y el importe y puntos de esta? ¿Influyen en ellos otras variables como el tipo de multa o el exceso de velocidad?
- ¿Qué denunciantes ponen más multas graves/muy graves?
- ¿Cada denunciante está relacionado con tipos de multas concretas?
- ¿En qué franjas horarias hay mayor número de multas graves y muy graves?
- ¿Qué tipos de multas graves/muy graves predominan?
- ¿Influye 'hecho' (tipo de multa) en la calificación y la sanción (importe y puntos) de las multas?
- ¿Qué tramos de exceso de velocidad son más frecuentes?
- ¿Influye el exceso de velocidad en la calificación y la sanción (importe y puntos) de las multas?
- ¿En qué lugares hay más multas graves/muy graves?
- ¿Hay lugares con más volumen de un tipo de multa grave/muy grave concreta?
- ¿Influye 'lugar' en la calificación y la sanción (importe y puntos) de las multas?

Se confecciona un esquema para tener una idea aproximada de las posibles relaciones entre variables.

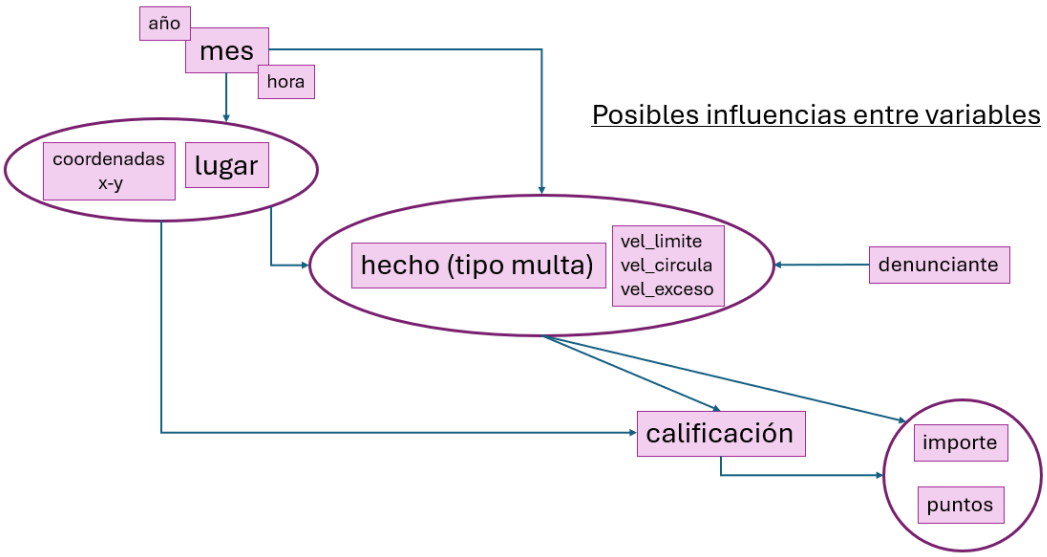


Ilustración 6: Esquema de posibles influencias entre variables

Y con esto en mente, ya podía dar comienzo al análisis exploratorio. Para el mismo, se tuvieron en cuenta los meses correspondientes a los años 2022 y 2023.

3.4.3. Análisis exploratorio

- Columna ‘calificacion’

Campo con 3 categorías: leve, grave y muy grave. Se creó una gráfica de barras con matplotlib para ver la distribución.

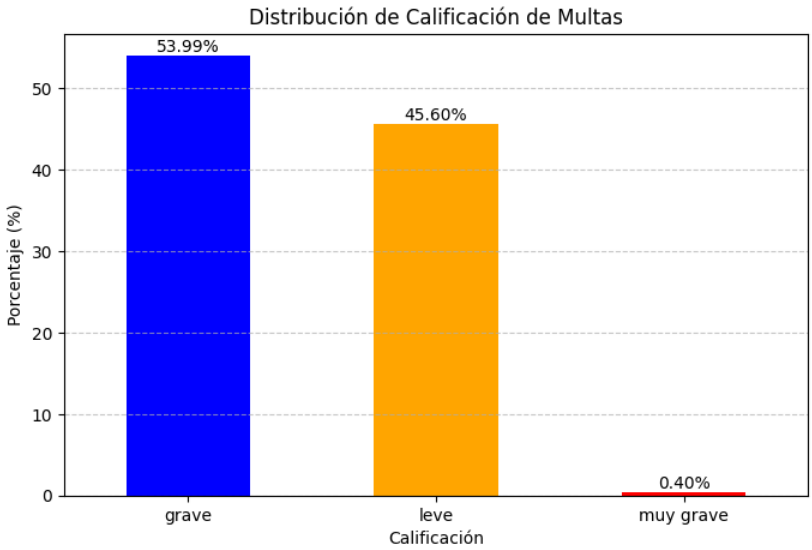


Ilustración 7: Distribución del campo 'calificacion'

Como se observa en la imagen, las multas calificadas como grave y leve tienen un volumen similar, sin embargo, el volumen de las calificadas como muy grave es muy pequeño, inferior al 1%. Por este motivo, de cara a las predicciones del modelo de ML, se decidió que la variable objetivo sería la suma de graves y muy graves.

- Columnas 'mes' y 'anio'

Para ver cómo se distribuyen las multas graves y muy graves en el tiempo, se creó un gráfico de líneas con matplotlib cuyo eje X es la conversión a fecha a través de la función de pandas to_datetime() y el eje Y la cantidad de multas graves y muy graves.

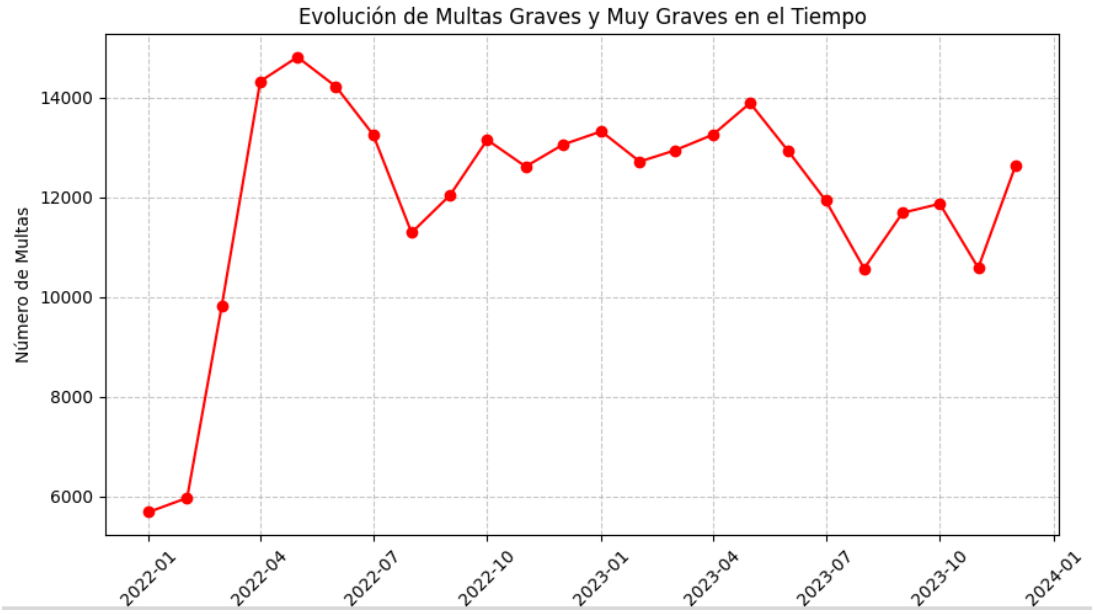


Ilustración 8: Evolución de multas graves y muy graves

Salvando la anomalía evidente de los meses enero y febrero de 2022, se aprecia un patrón: las multas aumentan en los meses abril y mayo y luego descienden hasta el mes de agosto, cuando vuelven a subir. Tiene bastante sentido, puesto que en los meses de junio, julio y agosto los ciudadanos de Madrid suelen abandonar el municipio por vacaciones, y los meses de abril y mayo, entre la Semana Santa y el puente del 1-2 de mayo, es probable que se desplacen más con sus vehículos, traduciéndose en un aumento en las multas.

- Columna ‘importe’

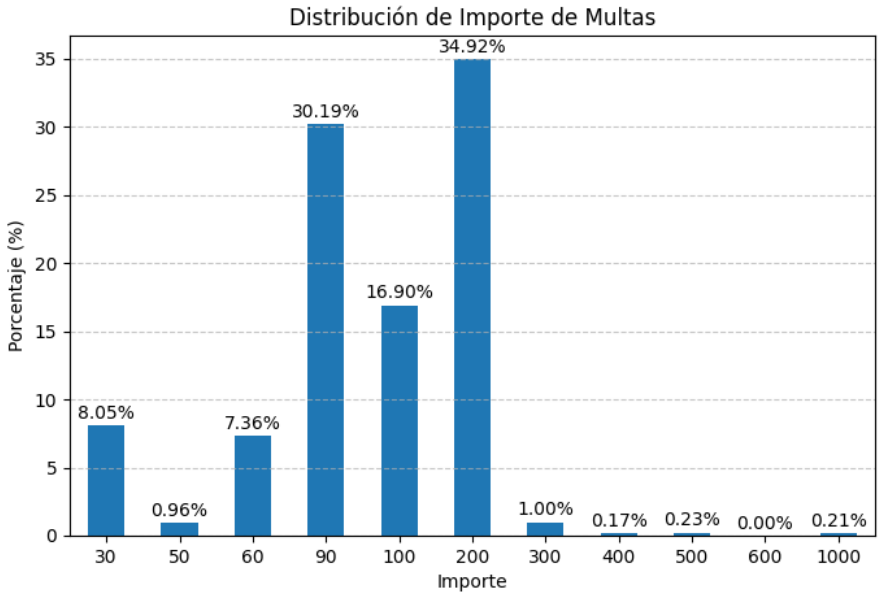


Ilustración 9: Distribución del campo ‘importe’

Campo con 11 categorías. Los importes más frecuentes son 90 y 200€, que suponen un 65,11% del total, y los menos frecuentes los importes igual o superior a 300€, con un volumen inferior al 2%.

Para analizar su relación con la calificación de las multas, se creó un boxplot con matplotlib y seaborn.

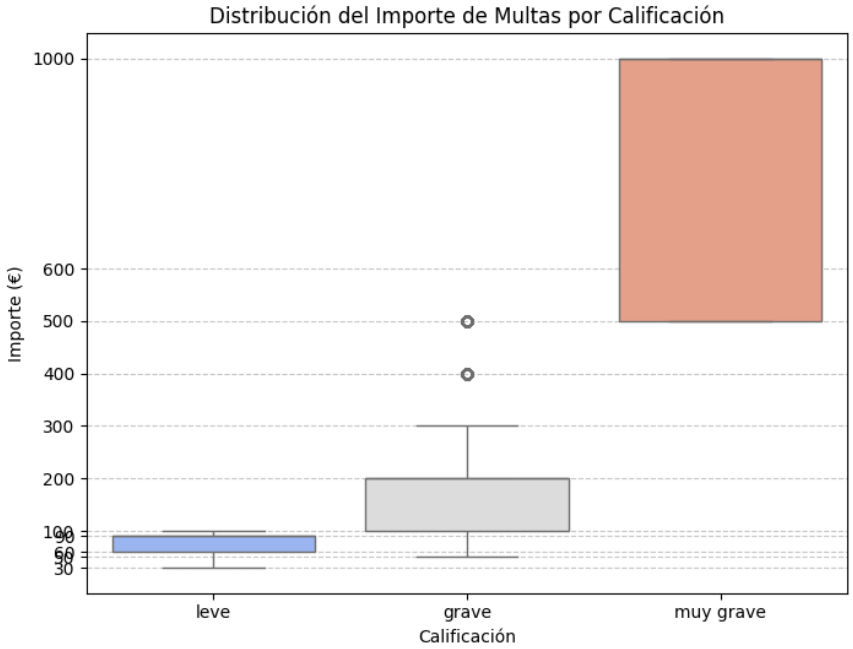


Ilustración 10: Distribución del campo 'importe' por 'calificacion'

Se observa claramente la relación directa, a mayor gravedad, mayor importe. El grueso de las multas leves son por los importes 60 y 90€, pero oscilan entre los 30 y los 100€, como marcan sus "bigotes". Las graves oscilan entre 50 y 300€, siendo lo habitual 100 o 200€. Hay dos outliers para esta calificación, es decir, dos valores atípicos que se desvían de la tendencia general de los datos. Por ejemplo, el importe de multa de 500€, rara vez tiene lugar en las multas graves, pero es muy frecuente en las muy graves, que se distribuyen de manera uniforme en torno a los valores 500 a 1000€.

En conclusión, el importe no solo está determinado por la gravedad de la multa, sino también por otras variables, que probablemente sean el motivo de la multa (columna 'hecho'), la cantidad de velocidad excedida (columna 'vel_exceso') o si se acumulan varias infracciones en el mismo hecho, por ejemplo, que el denunciado sobrepase la velocidad límite y además no tenga el permiso de conducir o la ITV en regla.

- Columna 'puntos'

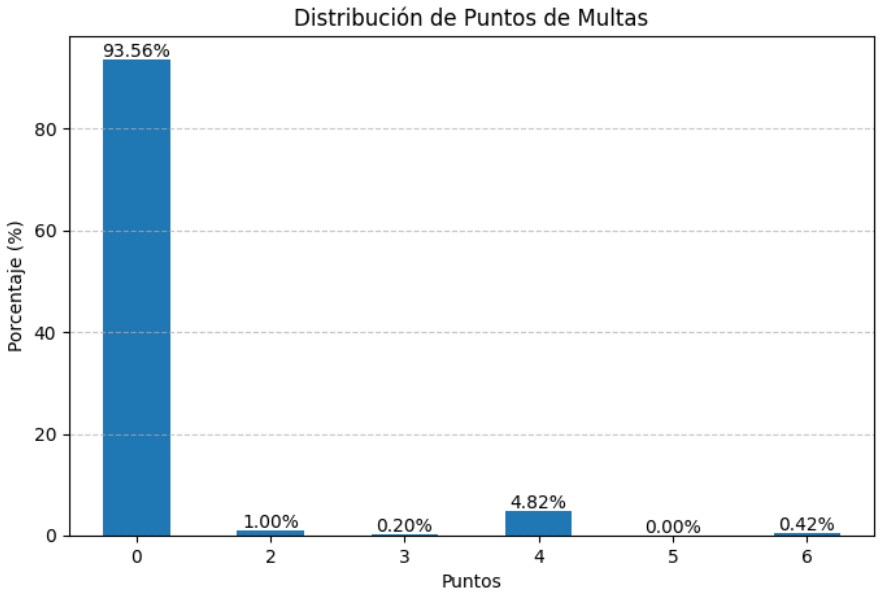


Ilustración 11: Distribución del campo 'puntos'

Campo con 6 categorías. Como se observa en la imagen, en la gran mayoría (93,56%) no hay retirada de puntos, en el 4,82% de las multas se retiran 4 puntos y en el resto de las categorías el volumen es inferior al 2%.

Está estrechamente relacionada con 'importe', ya que ambos campos suponen la sanción impuesta por la infracción cometida. Se creó también un boxplot:

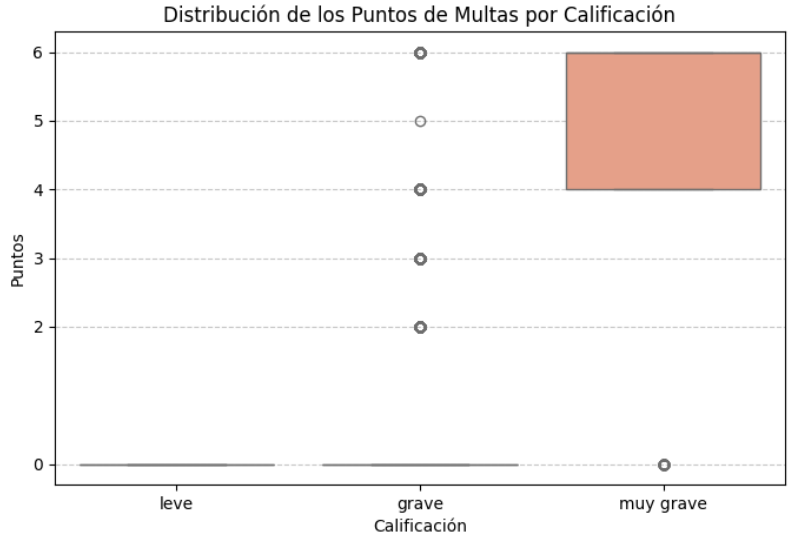


Ilustración 12: Distribución del campo 'puntos' por 'calificacion'

En este caso, las multas leves nunca conllevan retirada de puntos, las graves rara vez conllevan retirada de puntos, pero puede darse el caso, y las muy graves, al contrario, rara vez no conllevan retirada de puntos, que muy bien podría tratarse de usuarios a los que no le quedan puntos por retirar. Aunque aquí la estructura está más marcada que en 'importe', también deben influir otras variables como 'hecho' o 'vel_exceso'.

- Columna ‘denunciante’

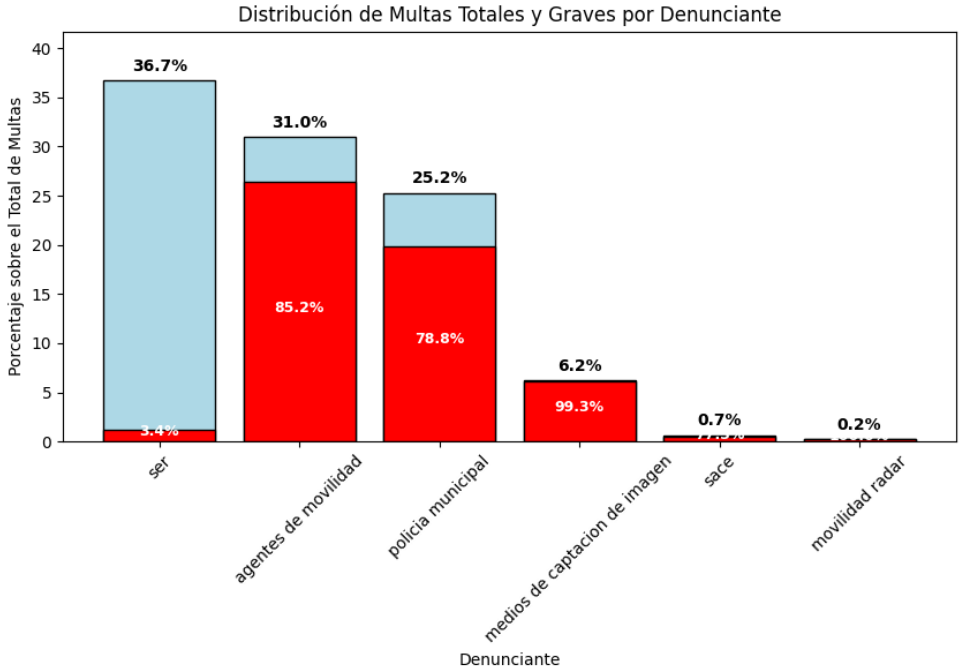


Ilustración 13: Distribución de multas por ‘denunciante’

Se trata de un campo con 6 categorías. Las categorías con mayor volumen total de multas son SER, Agentes de Movilidad y Policía Municipal y las que tienen menos son Medios de Captación de Imagen, Sace y Movilidad Radar. En cuanto a las multas graves y muy graves, Movilidad Radar sólo tiene esta calificación, Ser es el que tiene un porcentaje más pequeño (3,4%) y el resto de las categorías tienen un porcentaje alto, superior al 70%.

Se creó un boxplot para visualizar la relación del denunciante con los importes de multas graves y muy graves:

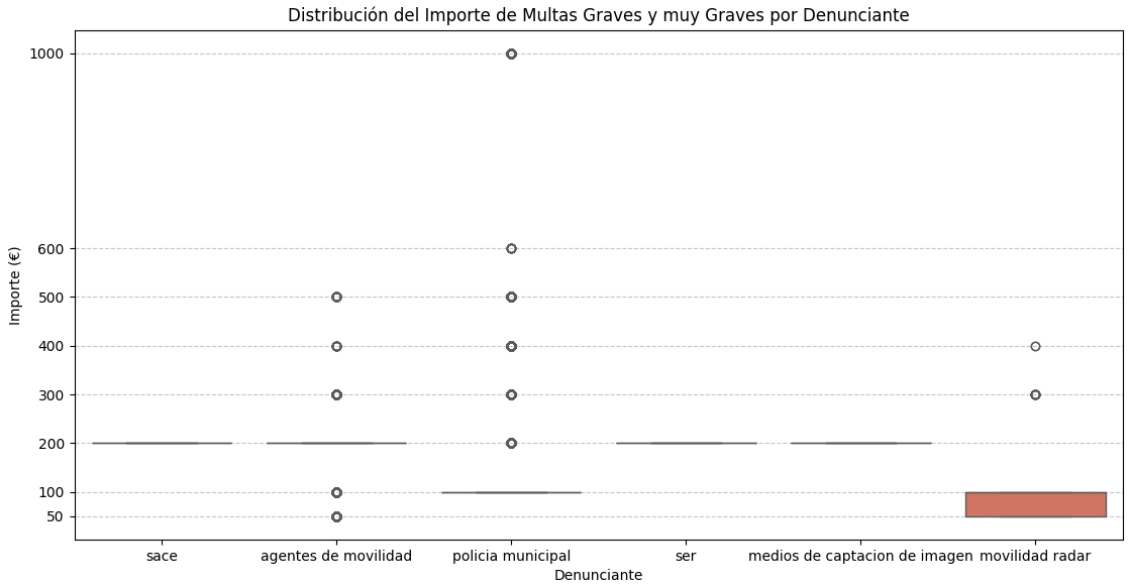


Ilustración 14: Distribución del campo 'importe' por 'denunciante' (multas graves/muy graves)

Algunos denunciantes, como Sace, Ser y Medios de Captación de Imagen, solo ponen multas graves y muy graves de un importe concreto (200€). Esto sugiere que cada denunciante puede poner multas de un tipo concreto y por lo tanto tendrían una relación directa con la columna 'hecho', y a su vez, es el hecho en sí lo que determina la sanción (multa y puntos).

Para corroborarlo se imprimen los 5 primeros valores únicos de 'hecho' en multas graves y muy graves para esos tres denunciantes que sólo registran importes de 200€:

```
Sace:
['estacionar en doble fila.'
 'estacionar en carril reservado para uso, parada o estacionamiento exclusivo del transporte público urbano.'
 'estacionar sobre la acera obstaculizando gravemente el tránsito de peatones.'
 'estacionar obstaculizando el giro del autobús.'
 'estacionar en intersección obstaculizando gravemente lacirculación.']

Ser:
['estacionar en doble fila.'
 'estacionar en zona señalizada para uso exclusivo de personas con movilidad reducida.'
 'estacionar en intersección obstaculizando gravemente lacirculación.'
 'estacionar obstaculizando la utilización de un vado señalizado correctamente.'
 'estacionar obstaculizando la utilización normal de unpaso de peatones.']

Medios de captación de imagen:
['acceder a la zbedep distrito centro sin autorización'
 'acceder a la zbedep plaza elíptica sin autorización'
 'rebasar un semáforo en fase roja.'
 'acceder a madrid zbe sin autorización']
```

Ilustración 15: Resultado de .unique()[:5] para 'hecho' por 'denunciante'

Efectivamente, tanto Sace como Ser ponen multas relacionadas con el estacionamiento indebido, y Medios de Captación de Imagen parece controlar las zonas de Madrid que requieren autorización o el control del tráfico.

- Columna ‘lugar’

Como ya se pudo ver en el análisis inicial, se trata de un campo muy complicado, con más de 60.000 valores únicos, que hubo que tratar de manera especial para reducir al máximo su dimensionalidad y obtener solo los tipos y nombres de vías. Es probable que haya vías con más tráfico que acumulen más multas u otras que por sus condiciones sean propensas a un tipo de multa concreto, pero se verá más adelante.

- Columna ‘hora’

Se asignaron franjas horarias para visualizar su distribución:

- Mañana: 6h a 14h
- Tarde: 14h a 22h
- Noche: 22h a 6h

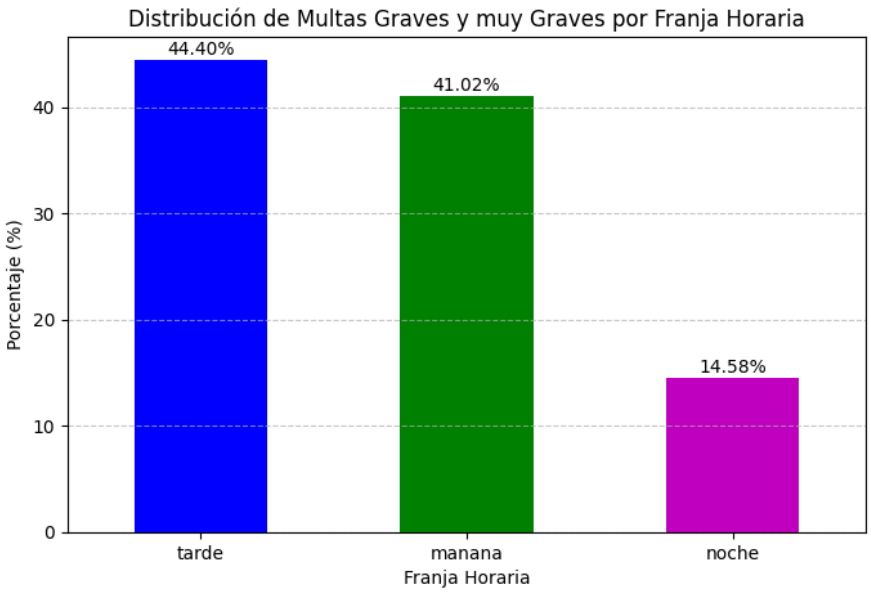


Ilustración 16: Distribución de multas graves/muy graves por franja horaria

Se observa que el volumen de multas graves y muy graves es muy similar por la mañana y por la tarde, y se reduce considerablemente por la noche.

A pesar de que la hora del día influya en el volumen de multas, el modelo que se ha creado realiza predicciones de meses completos, por lo que se prescindió de esta columna.

- Columna ‘hecho’

Se trata de otro campo complicado, con 587 valores únicos y cadenas largas donde se explica el motivo de la multa. Fueron necesarias técnicas de NLP para simplificar este

campo lo máximo posible. El motivo de la multa determina la calificación y, muy probablemente influye en la sanción, importe y puntos, pero se verá también más adelante.

- Columna ‘vel_exceso’

Se trata del resultado de restar ‘vel_limite’ a ‘vel_circula’. Se desglosó el campo en tramos de exceso de velocidad para visualizar la distribución en un gráfico de barras:

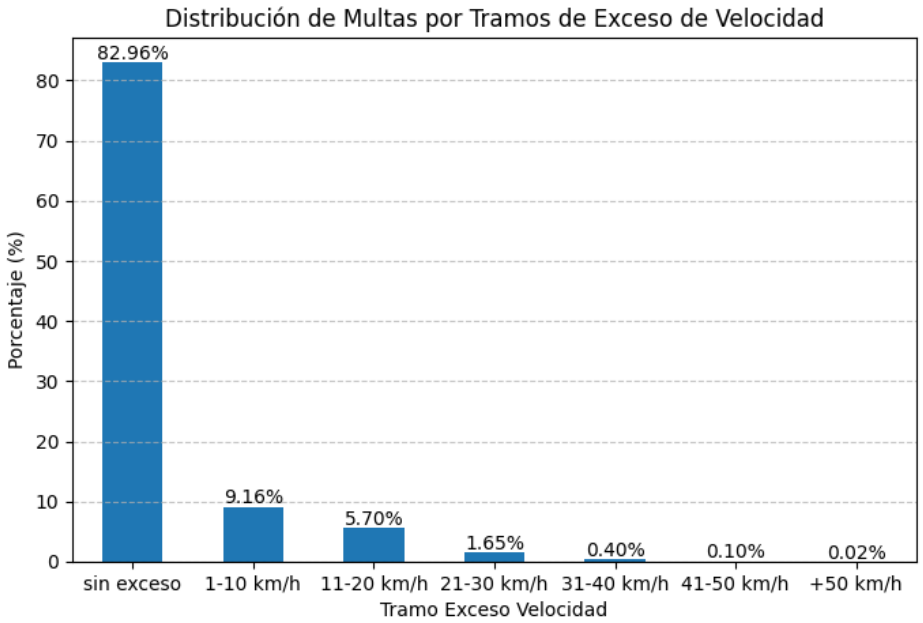


Ilustración 17: Distribución de multas por tramos de exceso de velocidad

La mayoría de los valores estaban en blanco, ya que este campo sólo se registra cuando el motivo de la multa es haber superado el límite de velocidad de la vía. El volumen de los tramos de exceso disminuye proporcionalmente al aumento del exceso de velocidad.

Se generó un gráfico de dispersión con matplotlib y seaborn para visualizar la relación entre el exceso de velocidad y el importe de la multa:

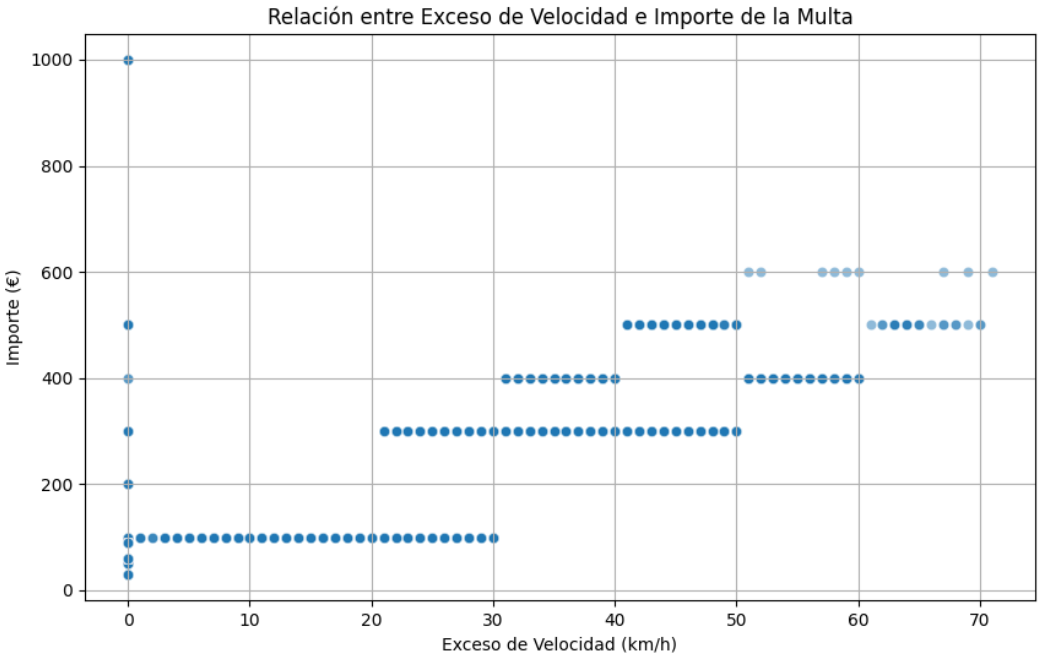


Ilustración 18: Relación entre 'vel_exceso' e 'importe'

La relación es clara, a mayor exceso de velocidad, mayor es el importe de la multa, aunque también hay valores con el mismo exceso que tienen importes diferentes.

Al filtrar el dataset por uno de los valores de 'vel_exceso' que tienen importes de multa distintos, por ejemplo 40, e imprimir los valores únicos de 'hecho', se obtuvo la respuesta:

```
Valores únicos de 'hecho' para 'vel_exceso' igual a 40:
['sobrepasar la velocidad máxima en vías limitadas en 60 km/h o más.'
 'sobrepasar la velocidad máxima en vías limitadas hasta 50 km/h.']
```

Ilustración 19: Valores únicos de 'hecho' para 'vel_exceso' igual a 40

No es el mismo importe de sanción un exceso de velocidad de 40 km/h en vías limitadas a 50 km/h que en vías limitadas a 60 km/h. Por lo tanto, 'vel_limite' también influye en el importe de la multa, además de 'hecho' e incluso 'lugar'.

Por este motivo, se mantuvieron las columnas 'vel_limite' y 'vel_exceso', pero se descartó la columna 'vel_circula' que no aportaba nada tras haber creado 'vel_exceso'.

- Columnas 'coordenada-x' y 'coordenada-y'

Según se indica en el Portal de datos de donde se obtuvo el dataset, estos campos de momento sólo se registran con ciertos tipos de multa. Se calculan los porcentajes de valores en blanco:

	Columna	Valores en blanco (%)	Valores no en blanco (%)
0	coordenada-x	65.23741	34.76259
1	coordenada-y	65.23741	34.76259

Ilustración 20: Porcentajes de valores en blanco de 'coordenada-x' y 'coordenada-y'

Se observa que la mayoría de los valores están en blanco. Además, tener en cuenta las coordenadas habría aumentado la dimensión de las vías, ya que marcan un punto concreto en el mapa dando lugar a múltiples coordenadas para una sola vía, y lo que interesaba para este proyecto es lo contrario, simplificar al máximo la columna 'lugar'. Es cierto que hubiesen sido interesantes para un análisis geoespacial, pero disponiendo solo de este dato en unos tipos concretos de multa, no tendría tenido sentido. En consecuencia, se descartaron ambas columnas.

3.5. Limpieza y transformación de los datos

3.5.1. Limpieza del dataset

Tras el análisis exploratorio se realizaron las siguientes modificaciones en el dataset:

- Eliminación de columnas innecesarias: 'hora', 'vel_circula', 'coordenada-x', 'coordenada-y'.
- Eliminación de los acentos de todas las columnas tipo texto con la librería unicode.

El siguiente paso fue preparar la columna 'lugar' para su modelado. Para la columna 'hecho' no fue necesario ya que se utilizó la librería SBERT que analiza las frases en su contexto natural.

3.5.2. Transformación de columna 'lugar'

En el mismo Portal de datos de donde se descargaron los datos de multas de circulación, se obtuvo también un callejero de Madrid con los viales vigentes.

Se llegó a la conclusión de que lo mejor sería usar una biblioteca de fuzzy matching (coincidencia difusa) para comparar la columna 'lugar' con el callejero y que devolviese una nueva columna llamada 'via' con el tipo y nombre de vía tal y como consta en el callejero.

Para ello, primero hubo que transformar la columna 'lugar' con el fin de facilitar al máximo la localización de coincidencias.

Llegados a este punto, para hacer una exploración amplia de los valores de la columna 'lugar' y saber qué transformaciones eran necesarias, resultó muy limitante trabajar en Python. Para esta tarea se implementó en el proyecto el uso de la

herramienta Microsoft Power BI, que permitió cargar el dataset completo de manera muy rápida y montar una tabla con los volúmenes y porcentajes de cada valor de 'lugar'. Así se pudieron comprobar de un vistazo los lugares con mayor volumen y qué modificaciones había que realizar sobre las cadenas para facilitar posteriormente la búsqueda de coincidencias con el callejero.

EXPLORACIÓN COLUMNA LUGAR

LUGAR	Recuento	%TG
CTRA. A42 PK 4.1 SENTIDO CREC.	162113	3,06%
CALLE ALCALA 51	158733	2,99%
CTRA. A42 PK 3.9 SENTIDO DECR.	107686	2,03%
CALLE GRAN VIA 71	106016	2,00%
CALLE ATOCHA 125	95418	1,80%
CL SEGOVIA ESQ RONDA SEGOVIA	71045	1,34%
CTRA. A42 PK 4.4 SENTIDO DECR.	68544	1,29%
CALLE SAN BERNARDO 91	66218	1,25%
TRAMO A5 Kms 4-5.7 SALIDA	61733	1,16%
M-30 CALZADA 2 KM 27.000	59200	1,12%
F005 AV PUERTA DE HIERRO	52144	0,98%
M-30 FT K 04	49201	0,93%
TRAMO A5 Kms 5.7-4 ENTRADA	45647	0,86%
CALLE EMBAJADORES 53	43358	0,82%
C. BARBARA DE BRAGANZA 13	41966	0,79%
A5 KM 4.0 SALIDA	39899	0,75%
CALLE TOLEDO 123	37974	0,72%
M-30 XC K 10	36942	0,70%
CALLE LEGANITOS 47	36210	0,68%
CALLE VALENCIA 25	34374	0,65%
PLAZA SANTA BARBARA 3	34340	0,65%
C. SANTA CRUZ DE MARCENADO 25	32709	0,62%
M-30 FT K 04 150 CR4	31397	0,59%
Total	**5305794**	**100,00%**

Ilustración 21: Exploración del campo 'lugar'

Tras las exploraciones, se llegó a la conclusión de que había que realizar las siguientes modificaciones:

- Eliminación de las filas en las que las cadenas no contienen ninguna letra.

```
# Eliminar filas en las que 'lugar' no contiene ninguna letra
multas = multas[multas["lugar"].str.contains(r"[a-z]", na=False)]
```

Código 5: Eliminación de filas sin letras en el campo 'lugar'

- Sustitución de los puntos ".", comas "," y guiones bajos "_" por espacios vacíos " ", ya que hay palabras unidas por ellos sin espacios y si se eliminan las palabras se juntarían.
- Eliminación de las barras "/" y los "º".
- Las autovías aparecen en la mayoría de casos en formato: "a 1", "a 2", "a 3", "a 4", "a 42", "a 5", "a 6", "m 11", "m 12", "m 13", "m 14", "m 21", "m 22", "m 30", "m 40", "m 45", "m 50", "m 500", "m 502", "m 503", "m 607", "r 2", "r 3", "r 5", "a-1", "a-2", "a-3", "a-4", "a-42", "a-5", "a-6", "m-11", "m-12", "m-13", "m-14", "m-21", "m-22", "m-30", "m-40", "m-45", "m-50", "m-500", "m-502", "m-503", "m-607", "r-2", "r-3" y "r-5". Dejarlos con los espacios dificultaría la labor

del fuzzy matching, y no se pueden dejar con los guiones puesto que se pretendía usar el guion como delimitador para eliminar la parte a la derecha de cada cadena en los casos en los que aparece una segunda vía para marcar la altura a la que se produce el hecho (como se comentó en la exploración inicial). Por lo tanto, se decide modificar estos patrones para que no tengan ni espacio " " ni guion "-" entre la letra y el dígito.

- Limpieza del resto de guiones "-": uso de .split() para obtener solo la parte de cadena anterior al guion.
- Se observa que después de la autovía suele haber información sobre kilómetro, salida, sentido, etc. que no aporta valor, así que se decide modificar todas las cadenas que contienen los patrones de autovías y autopistas limpiados anteriormente ("a1", "a5", "m30" …) y borrar el resto de cadena desde el patrón hacia la derecha.
- Eliminación las cadenas que suceden a la palabra "esq", por ejemplo, en el lugar "cl Segovia esq ronda Segovia", se elimina lo que sucede a "esq" que es información adicional que no interesa, además de eliminar la palabra "esq".
- Eliminación de los dígitos sueltos (números de las calles, kilómetros, etc.).
- Eliminación de palabras sueltas innecesarias: "de", "del", "de la", "de los", "de las", "al", "a la", "sn", "snn", "km", "pk", "fte", "esq", "sentido", "salida", "tramo", "farola", "túnel", "parking".
- Eliminación de unos patrones que aparecen con frecuencia, la letra k, f o n seguida de uno o varios dígitos: k970, f159, f011, n140, n034, etc.
- Eliminación de posibles espacios vacíos de más que hayan podido quedar.

Para las transformaciones se utiliza la biblioteca re, que trabaja muy bien con expresiones regulares a la hora de buscar, extraer y manipular texto.

```python
# Función de transformación
def transformar_lugar(lugar):
    """Función para aplicar las transformaciones previas al fuzzy matchching"""

    # Sustituir puntos, comas y guiones bajos por espacios vacíos
    lugar = re.sub(r"[.,_]", " ", lugar)

    # Eliminar los "/"y los "º"
    lugar = re.sub(r"[/º]", "", lugar)

    # Modificar patrones de autopistas y autovías
    patrones = [
    r"\ba 1\b", r"\ba 2\b", r"\ba 3\b", r"\ba 4\b", r"\ba 42\b", r"\ba 5\b",
    r"\ba 6\b", r"\bm 11\b", r"\bm 12\b", r"\bm 13\b", r"\bm 14\b",
    r"\bm 21\b", r"\bm 22\b", r"\bm 30\b", r"\bm 40\b", r"\bm 45\b",
    r"\bm 50\b", r"\bm 500\b", r"\bm 502\b", r"\bm 503\b", r"\bm 607\b",
    r"\br 2\b", r"\br 3\b", r"\br 5\b",
    r"\ba-1\b", r"\ba-2\b", r"\ba-3\b", r"\ba-4\b", r"\ba-42\b", r"\ba-5\b",
    r"\ba-6\b", r"\bm-11\b", r"\bm-12\b", r"\bm-13\b", r"\bm-14\b",
    r"\bm-21\b", r"\bm-22\b", r"\bm-30\b", r"\bm-40\b", r"\bm-45\b",
    r"\bm-50\b", r"\bm-500\b", r"\bm-502\b", r"\bm-503\b", r"\bm-607\b",
    r"\br-2\b", r"\br-3\b", r"\br-5\b"
    ]
    for patron in patrones:
        lugar = re.sub(
            patron, lambda match: match.group(0).replace(" ", "")
                                                .replace("-", ""), lugar
        )
```

Código 6: Transformaciones del campo 'lugar' (parte I)

```python
    # Limpiar el resto de guiones
    lugar = lugar.split("-")[0]

    # Borrar el resto de cadenas tras los patrones de autopistas/autovias
    lugar = re.sub(r"\b(a\d+|m\d+|r\d+)\b.*", r"\1", lugar)

    # Eliminar cadenas a la derecha de "esq"
    lugar = re.sub(r"\besq\b.*", "", lugar)

    # Eliminar números sueltos
    lugar = re.sub(r"(?<![-a-z])\b\d+\b(?![-a-z])", "", lugar)

    # Eliminar palabras innecesarias
    palabras = r"\b(de|del|de la|de los|de las|al|a la|sn|snn|" \
               r"km|pk|fte|esq|sentido|salida|tramo|farola|tunel|parking)\b"
    lugar = re.sub(palabras, "", lugar)

    # Eliminar palabras sueltas "k/f/n + dígitos" (k970, f159, n034...)
    lugar = re.sub(r"\b[kfn]\d+\b", "", lugar)

    # Eliminar espacios vacíos de más
    lugar = re.sub(r"\s+", " ", lugar).strip()

    return lugar

# Aplicar la función de transformación a la columna lugar
multas["lugar"] = multas["lugar"].apply(transformar_lugar)
```

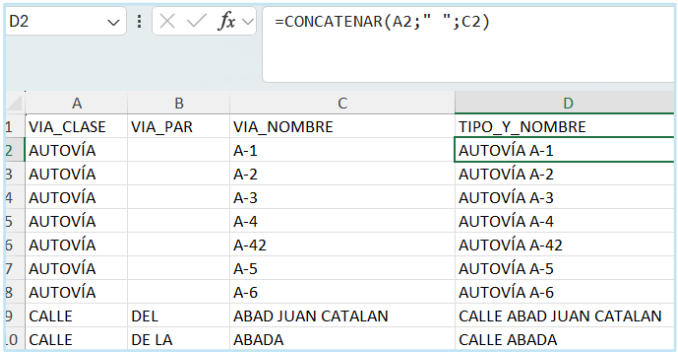
Código 7: Transformaciones del campo 'lugar' (parte II)

Tras comparar los valores de ‘lugar’ antes y después de estas transformaciones, en principio se determinó que el resultado era el adecuado.

3.5.3. Transformación del callejero

Primero se modificó el archivo directamente en Excel:

- Eliminación de todas las columnas excepto VIA_CLASE (tipo de vía), VIA_PAR (palabras que unen el tipo y el nombre de vía, si las hay) y VIA_NOMBRE.
- Eliminación de duplicados.
- Creación de una columna nueva llamada ‘TIPO_Y_NOMBRE’ con la concatenación de las columnas ‘VIA_CLASE’ y ‘VIA_NOMBRE’.



	A	B	C	D
1	VIA_CLASE	VIA_PAR	VIA_NOMBRE	TIPO_Y_NOMBRE
2	AUTOVÍA		A-1	AUTOVÍA A-1
3	AUTOVÍA		A-2	AUTOVÍA A-2
4	AUTOVÍA		A-3	AUTOVÍA A-3
5	AUTOVÍA		A-4	AUTOVÍA A-4
6	AUTOVÍA		A-42	AUTOVÍA A-42
7	AUTOVÍA		A-5	AUTOVÍA A-5
8	AUTOVÍA		A-6	AUTOVÍA A-6
9	CALLE	DEL	ABAD JUAN CATALAN	CALLE ABAD JUAN CATALAN
10	CALLE	DE LA	ABADA	CALLE ABADA

Ilustración 22: Ejemplo de campos del callejero

Después, se realizaron las siguientes modificaciones en Python:

- Eliminación de los guiones “-“ de las autopistas y autovías con re.
- Se observa que unas pocas vías del callejero también llevan guion “-“, por lo que se sustituyen por espacios vacíos.
- Transformación de encabezados y valores a minúsculas.
- Eliminación de acentos.
- Eliminación de las palabras sueltas “de”, “del”, “de la”, “de los”, “de las”, “al” y “a la” de los nombres de las vías, como se hizo en la columna ‘lugar’ del dataset, a fin de que se asemejen lo máximo posible.

Se guardó el resultado como un fichero CSV.

3.6. Modelado de los datos

3.6.1. Lugar: Fuzzy Matching

Para buscar coincidencias entre los valores de la columna 'lugar' y los valores de la columna 'tipo_y_nombre' del callejero y crear una nueva columna llamada "via" donde imputar las vías correctas, se decidió utilizar la librería RapidFuzz.

Esta librería dispone de cuatro tipos de cálculo:

- Proporción simple: fuzz.ratio() → Calcula la distancia de edición basándose en el orden de ambas cadenas de entrada.
- Proporción parcial: fuzz.partial_ratio() → Toma la cadena más corta y la compara con las subcadenas de la misma longitud de la cadena más larga. Serán parcialmente similares si tienen algunas de las palabras en un orden común.
- Token sort ratio: fuzz.token_sort_ratio() → Tiene en cuenta las cadenas similares sin importar en qué orden aparecen las palabras.
- Token set ratio: fuzz.token_set_ratio() → Parecido a token sort ratio, pero primero busca los tokens idénticos comunes en ambas cadenas y luego pasa a comparar las similitudes entre el resto de los tokens. Es útil es cadenas largas y cuando interesa encontrar tokens idénticos.

Se probó primero token_sort_ratio() y se anotaron algunos ejemplos de los resultados:

LUGAR	COINCIDENCIA ENCONTRADA	VÍA REAL QUE CORRESPONDE
01st csv	calle astun	ninguna coincidencia con callejero
a distribuidor torres 00ax82	calle estrecho torres	ninguna coincidencia con callejero
abalos post	calle dos caballos	calle abalos
abetal	calle abeto	calle abetal
abronigal glorieta m alvaro	glorieta angel caido	ninguna coincidencia con callejero
acceso plataforma alta merca	calle la picara molinera	ninguna coincidencia con callejero
acebeda	calle acebedo	calle la acebeda
aceuchal	calle acebuche	calle aceuchal
sainz bara	camino barrial	calle alcalde sainz de baranda

Ilustración 23: Resultados de RapidFuzz con token_sort_ratio()

Se observa que por defecto RapidFuzz siempre asigna una coincidencia a pesar de que no exista ninguna vía en el callejero que tenga sentido, por lo que hubo que establecer un límite para que no devolviese ninguna vía cuando la similitud fuese baja.

Por otro lado, token_sort_ratio() no funciona muy bien porque no busca la exactitud en los tokens sino la similitud.

La segunda prueba se realizó con token_set_ratio y se pudo comprobar que en este caso sí acierta con los lugares "abalos post", "abetal", "acebeda" y "aceuchal". Se explica a continuación la diferencia de procesamiento de ambas opciones con el lugar "abalos post":

Coincidencia de token_sort_ratio → calle dos caballos

Con token_sort_ratio elige "calle dos caballos" porque tiene 2 tokens similares (abalos similar a caballos, post similar a dos). En cambio, con "calle abalos" solo tiene 1 token en común (abalos), y tiene en consideración la exactitud de ese token.

Coincidencia de token_set_ratio → calle abalos

Con token_set_ratio elige “calle abalos” porque primero prioriza los tokens idénticos.

Se recopilaron más ejemplos para hacer una comparación de los resultados de los cuatro tipos de cálculo, añadiendo también una columna con el umbral de coincidencia (el porcentaje de similitud con la coincidencia encontrada).

Pero antes, tras las observaciones realizadas de los ejemplos recopilados y dado que para token_set_ratio es importante encontrar tokens idénticos, se realizó una segunda tanda de transformaciones en la columna ‘lugar’, también con la biblioteca re:

- Sustitución de las abreviaturas de los tipos de vías por la palabra completa:
    - "p" y "po" por "paseo"
    - "av" y "avda" por "avenida"
    - "ctra" y "cr" por "carretera"
    - "cl" y "c" por "calle"
    - "pz" y "pza" por "plaza"
    - "cu" y "cta" por "cuesta"
    - "ro" y "rda" por "ronda"
    - "gt" y "gta" por "glorieta"
    - "gv" por "gran via"
    - "tr" y "trav" por "travesia"
    - "pj" y "pje" por "pasaje"
    - "cra" por "carrera"
    - "pte" por "puente"
    - "cno" por "camino"
- Sustitución de todas las cadenas que contienen la palabra "aeropuerto" por "aeropuerto adolfo suarez madrid barajas", tal como viene en el callejero.
- Localización de las cadenas que empiezan por los patrones "a1", "a2", "a3", "a4", "a42", "a5", "a6", "m11", "m12", "m13", "m14", "m21", "m22", "m30", "m40", "m45", "m50", "m500", "m502", "m503", "m607" para añadir al inicio de la palabra "autovia".
- Localización de las cadenas que empiezan por los patrones "r2", "r3" y "r5" para añadir al inicio la palabra "autopista".
- Eliminación de todas las filas que quedaron con ‘lugar’ en blanco.

Tras estos cambios, se confeccionó una tabla de Excel con los resultados de los cuatro métodos de RapidFuzz para los ejemplos recopilados:

LUGAR		FUZZ.RATIO		FUZZ.PARTIAL_RATIO	
lugar dataset	vía callejero	coincidencia	puntaje	coincidencia	puntaje
abalos	calle abalos	calle abalos	66,666667	calle abalos	100
alto retiro feuvert	calle alto retiro	calle alto retiro	61,111111	calle alto retiro	78,571429
alto retiro carrefour	calle alto retiro	calle alto retiro	57,894737	calle alto retiro	78,571429
aracne bauhaus	calle aracne	calle acebuche	50	calle aracne	66,666667
avenida aguilas ctro salud	avenida las aguilas	avenida las aguilas	66,666667	avenida las aguilas	88,235294
avenida america f28	avenida america	avenida america	88,235294	avenida america	100
avenida burgos	avenida burgos	avenida burgos	100	avenida burgos	100
avenida central dhl	avenida central	avenida central	88,235294	avenida central	100
calle ayala 88a	calle ayala	calle ayala	84,615385	calle ayala	100
calle sarria 56 post	calle sarria	calle sarria	75	calle sarria	100
01st csv		calle dos caballos	30,769231	calle acueducto segovia	50
a distribuidor torres 00ax82		calle estrecho torres	53,061224	calle estrecho torres	61,904762
abades	calle abades	calle abades	66,666667	calle abades	100
abalos post	calle abalos	calle abalos	52,173913	calle abalos	70,588235
abetal	calle abetal	calle abetal	66,666667	calle abetal	100
abronigal glorieta m alvaro		calle mendez alvaro	56,521739	glorieta eolo	76,923077
acceso plataforma alta merca		plaza alhama almeria	54,166667	avenida america	64
acebeda	calle la acebeda	calle la acebeda	60,869565	calle la acebeda	100
aceuchal	calle aceuchal	calle aceuchal	72,727273	calle aceuchal	100
sainz bara	cale alcalde sainz baranda	camino barrial	66,666667	calle alcalde sainz baranda	100
carabela	calle carabela	calle carabela	72,727273	calle carabela	100
alcocer	calle alcocer	calle alcocer	70	avenida alberto alcocer	100
almeria	calle almeria	calle almeria	70	plaza alhama almeria	100
angel	calle angel	calle angel	62,5	glorieta angel caido	100
aragon	calle aragon	calle aragon	66,666667	calle agustina aragon	100
arevalo	calle arevalo	calle arevalo	70	glorieta alcalde canillas marco arevalo perez	100
arevalo lara calle	calle arevalo lara	calle arevalo lara	66,666667	calle g	83,333333
avenida aurora boreal calle futbol	avenida aurora boreal	avenida aurora boreal	76,363636	avenida aurora boreal	100
calle segovia	calle segovia	calle segovia	100	calle segovia	100
calle a5	autovia a5	calle g	80	calle abalos	93,333333
mendez alvaro m30	calle mendez alvaro	calle mendez alvaro	72,222222	plaza amanecer en mendez alvaro	86,666667
via servicio m30	autovia m30	autovia m30	51,851852	avenida america	64
abay	calle abay	calle abay	57,142857	calle abay	100
abubilla	ronda abubilla	ronda abubilla	72,727273	ronda abubilla	100

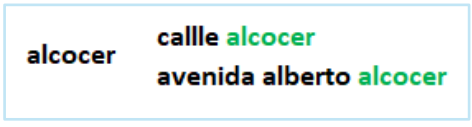
Ilustración 24: Comparativa de los resultados de los cuatro tipos de Fuzzy Matching (parte I)

- **Ratio**: Tan sólo devuelve 3 fallos, pero el umbral de similitud es bastante bajo en general, con un promedio de 67,87%. Esto obligaría a establecer un umbral bajo para obtener aciertos, corriendo también el riesgo de añadir equivocaciones.
- **Partial_ratio**: Devuelve 9 fallos con un promedio de umbral del 90,08%. La mayoría de los fallos devueltos tienen un umbral del 100%, suponiendo un grave riesgo para la fiabilidad de los resultados.

LUGAR		FUZZ.TOKEN_SORT_RATIO		FUZZ.TOKEN_SET_RATIO	
lugar dataset	vía callejero	coincidencia	puntaje	coincidencia	puntaje
abalos	calle abalos	calle abalos	66,666667	calle abalos	100
alto retiro feuvert	calle alto retiro	calle alto retiro	72,222222	calle alto retiro	78,571429
alto retiro carrefour	calle alto retiro	calle alto retiro	78,947368	calle alto retiro	78,947368
aracne bauhaus	calle aracne	calle aracne	61,538462	calle aracne	66,666667
avenida aguilas ctro salud	avenida las aguilas	avenida las aguilas	75,555556	avenida las aguilas	88,235294
avenida america f28	avenida america	avenida america	88,235294	avenida america	100
avenida burgos	avenida burgos	avenida burgos	100	avenida burgos	100
avenida central dhl	avenida central	avenida central	88,235294	avenida central	100
calle ayala 88a	calle ayala	calle ayala	84,615385	calle ayala	100
calle sarria 56 post	calle sarria	calle sarria	75	calle sarria	100
01st csv		calle astun	42,105263	calle astun	42,105263
a distribuidor torres 00ax82		calle estrecho torres	53,061224	calle estrecho torres	53,061224
abades	calle abades	calle abades	66,666667	calle abades	100
abalos post	calle abalos	calle dos caballos	62,068966	calle abalos	70,588235
abetal	calle abetal	calle abeto	70,588235	calle abetal	100
abronigal glorieta m alvaro		glorieta angel caido	68,085106	glorieta eolo	76,190476
acceso plataforma alta merca		calle la picara molinera	57,692308	calle la picara molinera	57,692308
acebeda	calle la acebeda	calle acebedo	70	calle la acebeda	100
aceuchal	calle aceuchal	calle acebuche	72,727273	calle aceuchal	100
sainz bara	cale alcalde sainz baranda	camino barrial	66,666667	camino barrial	66,666667
carabela	calle carabela	calle carabela	72,727273	calle carabela	100
alcocer	calle alcocer	calle alcocer	70	avenida alberto alcocer	100
almeria	calle almeria	calle almeria	70	plaza alhama almeria	100
angel	calle angel	calle angel	62,5	glorieta angel caido	100
aragon	calle aragon	calle aragon	66,666667	calle agustina aragon	100
arevalo	calle arevalo	calle arevalo	70	glorieta alcalde canillas marco arevalo perez	100
arevalo lara calle	calle arevalo lara	calle arevalo lara	100	calle arevalo	100
avenida aurora boreal calle futbol	avenida aurora boreal	avenida aurora boreal	76,363636	calle aurora	100
calle segovia	calle segovia	calle segovia	100	calle acueducto segovia	100
calle a5	autovia a5	calle abay	77,777778	calle g	83,333333
mendez alvaro m30	calle mendez alvaro	calle mendez alvaro	77,777778	plaza amanecer en mendez alvaro	86,666667
via servicio m30	autovia m30	avenida america	58,064516	autovia m30	59,259259
abay	calle abay	calle abay	57,142857	calle abay	100
abubilla	ronda abubilla	ronda abubilla	72,727273	ronda abubilla	100

Ilustración 25: Comparativa de los resultados de los cuatro tipos de Fuzzy Matching (parte II)

- **Token_sort_ratio**: Devuelve 7 fallos y un promedio de umbral del 72,13%. Con esta opción surge el problema de la similitud de tokens comentada anteriormente en el ejemplo “abalos post”.
- **Token_set_ratio**: Devuelve 11 fallos y un promedio de umbral del 88,47%. Al revisar los fallos, llama la atención que, por ejemplo, para el lugar “alcocer” elige “avenida alberto alcocer” en lugar de “calle alcocer”. El umbral de similitud es del 100%, por lo que se elimina la vía “avenida alberto alcocer” de la prueba para ver qué umbral obtiene “calle alcocer”, dando como resultado también el 100%. Esto ocurre porque ambas opciones tienen un token exacto, empatando en el puntaje de similitud obtenido.
  Tras buscar en el manual de RapidFuzz qué opción escoge al encontrar varias coincidencias con el mismo umbral de similitud, se verifica que simplemente elige el valor que va primero en el orden de la lista original.



Para solucionarlo, se implementó un modelo mixto token_set_ratio / ratio_simple. En lugar de usar el módulo process.extractOne() para extraer solo una coincidencia, se usa primero el módulo process.extract() con token_set_ratio para obtener una lista de todas las coincidencias y obtener aquella o aquellas con el puntaje máximo. Después, si este puntaje máximo no supera el umbral de similitud establecido, devuelve un valor en blanco, y si supera el umbral pueden ocurrir 2 cosas:

- Que solo haya una coincidencia con el puntaje más alto, y entonces será la coincidencia final (de token_set_ratio).
- Que haya varias coincidencias con el puntaje más alto, y entonces usará ratio simple para obtener la mejor coincidencia basada en los edits en lugar de tokens exactos.

Se ejecutó la prueba de este modelo mixto con los ejemplos recopilados y el resultado en principio fue bueno, devolviendo 2 fallos con un promedio de umbral (de token_set_ratio) de 88,47%.

LUGAR		TOKEN_SET_RATIO DESEMPATE CON RATIO	
lugar dataset	vía callejero	coincidencia	puntaje
abalos	calle abalos	calle abalos	100
alto retiro feuvert	calle alto retiro	calle alto retiro	78,571429
alto retiro carrefour	calle alto retiro	calle alto retiro	78,947368
aracne bauhaus	calle aracne	calle aracne	66,666667
avenida aguilas ctro salud	avenida las aguilas	avenida las aguilas	88,235294
avenida america f28	avenida america	avenida america	100
avenida burgos	avenida burgos	avenida burgos	100
avenida central dhl	avenida central	avenida central	100
calle ayala 88a	calle ayala	calle ayala	100
calle sarria 56 post	calle sarria	calle sarria	100
01st csv		calle astun	42,105263
a distribuidor torres 00ax82		calle estrecho torres	53,061224
abades	calle abades	calle abades	100
abalos post	calle abalos	calle abalos	70,588235
abetal	calle abetal	calle abetal	100
abronigal glorieta m alvaro		glorieta eolo	76,190476
acceso plataforma alta merca		calle la picara molinera	57,692308
acebeda	calle la acebeda	calle la acebeda	100
aceuchal	calle aceuchal	calle aceuchal	100
sainz bara	cale alcalde sainz baranda	camino barrial	66,666667
carabela	calle carabela	calle carabela	100
alcocer	calle alcocer	calle alcocer	100 / 70
almeria	calle almeria	calle almeria	100 / 70
angel	calle angel	calle angel	100 / 62,5
aragon	calle aragon	calle aragon	100 / 66,67
arevalo	calle arevalo	calle arevalo	100 / 70
arevalo lara calle	calle arevalo lara	calle arevalo lara	100 / 66,67
avenida aurora boreal calle futbol	avenida aurora boreal	avenida aurora boreal	100 / 76,36
calle segovia	calle segovia	calle segovia	100 / 100
calle a5	autovia a5	calle g	83,333333
mendez alvaro m30	calle mendez alvaro	calle mendez alvaro	83,33 / 72,22
via servicio m30	autovia m30	autovia m30	59,259259
abay	calle abay	calle abay	100
abubilla	ronda abubilla	ronda abubilla	100

Ilustración 26: Resultado del modelo RapidFuzz mixto (token_set_ratio / ratio_simple)

En cuanto al umbral de similitud a establecer para discriminar coincidencias, en base a los ejemplos recopilados se determinó que un umbral equilibrado podría ser 78%, que tan sólo deja 1 vía errónea de las 6 existentes, y elimina 3 vías correctas. Establecerlo más alto hubiese supuesto eliminar más buenas que malas, y más bajo hubiese supuesto dejar más vías erróneas. Como se dispone de muchos datos, es preferible sacrificar filas del dataset a tener vías que no son correctas y que los resultados del proyecto no sean fiables.

No obstante, se realizó una última prueba ejecutando RapidFuzz en el dataset de multas con los parámetros mencionados y añadiendo la columna con el umbral de similitud para examinar los resultados y comprobar si era necesario algún ajuste más.

```python
# Crear una lista con las vías
lista_vias = callejero['tipo_y_nombre'].tolist()

# Función para encontrar la coincidencia más cercana
def match_lugar(lugar, lista_vias, umbral=78):
    """
    Función que busca la mejor coincidencia entre lugar y callejero.
    Usa token_set_ratio con umbral de similitud superior o igual a 78.
    En caso de empate en similitud, desempata con ratio simple.
    """

    # Obtener todas las coincidencias con el puntaje más alto
    resultados = process.extract(
        lugar, lista_vias, scorer=fuzz.token_set_ratio, limit=None)
    max_puntaje = max(m[1] for m in resultados) if resultados else 0
    vias_mejor_puntaje = [m for m in resultados if m[1] == max_puntaje]
```

```python
    # Si no se supera el umbral, devuelve None (en blanco)
    if max_puntaje < umbral:
        return None, None

    # Si hay empate, usar ratio para desempatar
    if len(vias_mejor_puntaje) > 1:
        mejor_via = max(
            vias_mejor_puntaje, key=lambda m: fuzz.ratio(lugar, m[0]))
            # vía de ratio (mejor_via[0])
        puntaje_final = fuzz.ratio(
            lugar, mejor_via[0])  # % similitud de ratio
    else:
        mejor_via = vias_mejor_puntaje[0]  # vía de token_set_ratio
        puntaje_final = mejor_via[1]  # % similitud de token_set_ratio

    return mejor_via[0], puntaje_final # vía y similitud final

# Aplicar función de fuzzy matching
multas[['via', 'similitud']] = multas['lugar'].apply(
    lambda lugar: pd.Series(match_lugar(lugar, lista_vias))
)
```

Código 8: Ejecución del modelo RapidFuzz mixto (token_set_ratio / ratio_simple)

Tras revisar los datos obtenidos se concluyó que los resultados eran muy satisfactorios, aunque había que realizar todavía unos últimos ajustes en la columna ‘lugar’:

- Sustitución de "carretera a42" por "autovia a42".
- Sustitución de “sm” por "santa maria".
- Eliminación de “subtrraneo”.
- Eliminación de filas que hubiesen quedado con ‘via’ en blanco (discriminaciones del umbral de similitud).
- Eliminación de la columna ‘similitud’ que ya no es necesaria.

RESULTADOS RAPIDFUZZ

via / lugar	Recuento	%TG
⊞ autovia m30	56422	10,64%
⊞ autovia a42	33561	6,33%
⊞ calle alcala	19702	3,71%
⊞ autovia a5	14676	2,77%
⊞ calle gran via	11572	2,18%
⊞ calle atocha	9961	1,88%
⊞ paseo castellana	9287	1,75%
⊞ calle san bernardo	7268	1,37%
⊟ calle segovia	7258	1,37%
calle segovia	7210	1,36%
segovia	44	0,01%
calle segovia 47a	3	0,00%
calle segovia 47b	1	0,00%
⊞ calle embajadores	5888	1,11%
⊞ avenida puerta hierro	5663	1,07%
⊞ calle toledo	4559	0,86%
⊞ calle barbara braganza	4374	0,82%
⊞ paseo santa maria cabeza	4358	0,82%
⊞ calle leganitos	3791	0,71%
⊞ calle doctor esquerdo	3568	0,67%
⊞ calle valencia	3468	0,65%
⊞ calle santa cruz marcenado	3427	0,65%
⊞ plaza santa barbara	3424	0,65%
⊟	3411	0,64%
gral ricardos	1142	0,22%
carab aravaca	868	0,16%
f	297	0,06%
ayerbe mercadillo	69	0,01%
cjal benito mtin lozano 12b	43	0,01%
avenida la institucion libr	28	0,01%
Total	530433	100,00%

Ilustración 27: Resultado final de RapidFuzz

3.6.2. Hecho: Clusterización

Para formar clústeres a partir de la columna ‘hecho’ y crear así una nueva columna ‘tipo_multa’ con las tipologías simplificadas de los motivos de multa, se implementó una combinación de SBERT + UMAP + DBSCAN.

- **SBERT**: La clusterización no puede aplicarse directamente sobre texto, primero es necesario crear una representación numérica del texto para que DBSCAN pueda aplicar el algoritmo correspondiente y agrupar en clústeres.
  Se escogió SBERT porque usa representaciones vectoriales (embeddings) de textos cortos capturando el significado de las oraciones, a diferencia de BERT que genera embeddings de tokens individuales. Como la columna ‘hecho’ tiene muchos valores diferentes que en realidad son el mismo motivo de multa, pero con otras palabras, se consideró que SBERT funcionaría bien.
- **UMAP**: No es imprescindible reducir la dimensionalidad de los embeddings obtenidos con SBERT, se podría aplicar el clustering directamente, pero en

teoría hace que el procesamiento de DBSCAN seas más eficiente ya que puede acercar más entre sí los puntos parecidos y evitar el ruido (aquellos valores que no se agrupan en ningún clúster). De todos modos, se probó con y sin UMAP para verificar qué opción daba mejores resultados.
Se escoge UMAP porque preserva mejor la estructura local y global de los datos y funciona mejor con SBERT que otros algoritmos. Además, es bastante rápido.

- **DBSCAN**: Es el algoritmo que realiza el clustering propiamente dicho.
Se elige DBSCAN porque, a diferencia de K-Means, puede detectar grupos de cualquier forma, no solo esféricos. También marca el ruido como clúster -1, lo que puede ayudar a reducir el ruido realizando pruebas con diferentes parámetros. Además, no es necesario asignarle un número de clústeres ya que los detecta automáticamente.

El procesamiento se aplicó sobre los valores únicos de 'hecho', ya que sobre el total hubiese sido muy costoso computacionalmente al generar los embeddings.

Hay varios parámetros principales que pueden configurarse, aunque se decide ajustar solo los más importantes:

- **n_components (UMAP)**: número de dimensiones a las que reducir los embeddings.
- **n_neighbors (UMAP)**: número de vecinos para formar clústeres. Los valores bajos obligarán a UMAP a concentrarse en una estructura muy local (potencialmente en detrimento del panorama general), mientras que los valores grandes empujarán a UMAP a buscar vecindarios más grandes de cada punto al estimar la estructura múltiple de los datos (perder la estructura de los detalles finos en aras de obtener la información más amplia).
- **min_dist (UMAP)**: controla la separación entre puntos. Valores bajos agrupa los puntos más juntos.
- **eps (DBSCAN)**: es la distancia máxima para que un punto sea considerado vecino. Un eps bajo generará más clústeres, aunque también puede aumentar el ruido.
- **min_samples (DBSCAN)**: número mínimo de puntos para formar un clúster. Si es bajo puede crear muchos clústeres pequeños o incluir ruido en los clústeres.

Para no realizar múltiples pruebas de hiperparámetros a ciegas, se utilizó primero la librería Optuna para optimizar los hiperparámetros mencionados y que el procesamiento devolviese menos de 60 clústeres con el menor ruido posible.

```python
# Función para optimizar UMAP + DBSCAN
def optimizar(test):
    # Hiperparámetros a optimizar
    n_components = test.suggest_int("n_components", 2, 50)
    n_neighbors = test.suggest_int("n_neighbors", 5, 50)
    min_dist = test.suggest_float("min_dist", 0.0, 0.99)
    eps = test.suggest_float("eps", 0.1, 5.0)
    min_samples = test.suggest_int("min_samples", 2, 20)

    # Reducir dimensionalidad con UMAP
    reducer = umap.UMAP(
        n_components=n_components,
        n_neighbors=n_neighbors,
        min_dist=min_dist,
        random_state=42)
    reduced_embeddings = reducer.fit_transform(embeddings)

    # Aplicar DBSCAN
    clustering = DBSCAN(
        eps=eps, min_samples=min_samples).fit(reduced_embeddings)
    labels = clustering.labels_

    # Contar ruido (-1) y número de clusters
    ruido = np.sum(labels == -1)
    num_clusters = len(set(labels)) - (1 if -1 in labels else 0)

    # Penalización si se exceden 60 clusters
    if num_clusters > 60:
        return ruido + (num_clusters - 60) * 10  # Penalización fuerte
    return ruido

# Ejecutar optimización
study = optuna.create_study(direction="minimize")
study.optimize(optimizar, n_trials=50)
```

Código 9: Optimización de hiperparámetros con Optuna para UMAP y DBSCAN

```
Mejores hiperparámetros encontrados:
n_components: 24
n_neighbors: 17
min_dist: 0.9165600827178009
eps: 2.233325269179569
min_samples: 3
```

Ilustración 28: Resultados de optimización de hiperparámetros para UMAP y DBSCAN

Con estos ajustes ya se obtuvieron unos clústeres decentes, pero con bastantes valores de ruido (-1), así que se fueron realizando ajustes de hiperparámetros manualmente arriba y abajo hasta encontrar una combinación más o menos aceptable. También se probó sin UMAP, pero los resultados fueron peores. Aun así, se tuvieron que realizar agrupaciones manuales de clústeres parecidos y mover algunos valores a otros clústeres, ya que resultó muy complicado encontrar un equilibrio óptimo entre el número de clústeres y la coherencia de estos. Fue preferible obtener un mayor número de clústeres y después unir los que fuesen parecidos, que obtener clústeres más grandes que no tuviesen valores homogéneos. En principio se obtuvieron 53 clústeres.

El siguiente paso fue modificar el código para incluir los valores que resultaron en ruido (-1) a su clúster más cercano, y para ello se implementó el módulo NearestNeighbors de Scikit-learn. El código final quedó así:

```python
# Agrupar valores repetidos
recuento_hechos = Counter(multas['hecho'])
df_unicos = pd.DataFrame(
    recuento_hechos.items(), columns=['hecho', 'frecuencia'])
print("Agrupación completada.")

# Cargar modelo SBERT
model = SentenceTransformer('all-MiniLM-L6-v2')
print("Modelo de embeddings cargado.")

# Obtener embeddings
embeddings = model.encode(df_unicos['hecho'].tolist(), convert_to_numpy=True)
df_unicos['embedding'] = list(embeddings)  # Guardar como listas en el DataFrame
print("Obtención de embeddings completada.")

# Reducir dimensionalidad con UMAP
reducer = umap.UMAP(
    n_components=50,
    n_neighbors=5,
    min_dist=0.05,
    random_state=42,
)
X_reduced = reducer.fit_transform(embeddings)
print("Reducción de dimensionalidad completada.")
```

```python
# Aplicar DBSCAN
dbscan = DBSCAN(
    eps=0.3,  # Distancia máxima para ser considerado vecino
    min_samples=3  # Mínimo de muestras para formar un cluster
)
clusters_labels = dbscan.fit_predict(X_reduced)
df_unicos['cluster'] = clusters_labels
print("DBSCAN completado.")

# Asignar ruido (-1) al clúster más cercano
ruido = df_unicos['cluster'] == -1
if ruido.sum() > 0:

    # Entrenar modelo de vecinos solo con puntos no ruidosos
    X_limpio = X_reduced[df_unicos['cluster'] != -1]
    labels_limpios = clusters_labels[df_unicos['cluster'] != -1]

    neigh = NearestNeighbors(n_neighbors=1, metric='euclidean')
    neigh.fit(X_limpio)

    # Encontrar el vecino más cercano para cada punto ruidoso
    X_ruido = X_reduced[ruido]
    distancias, indices = neigh.kneighbors(X_ruido)

    # Asignar el clúster del vecino más cercano
    df_unicos.loc[ruido, 'cluster'] = labels_limpios[indices.flatten()]

print("Reasignación completada.")
```

Código 10: Procesamiento de clusterización con SBERT + UMAP + DBSCAN

Tras realizar un estudio del contenido de los clústeres obtenidos, se decidió mantener solo 18 clústeres definitivos:

1. **Estacionamiento indebido**: Estacionar o parar en zonas no permitidas o de manera inadecuada.
2. **Desobediencia a agentes**: No obedecer órdenes de los agentes de tráfico.
3. Uso indebido de dispositivos electrónicos: Uso de móviles, navegadores u otros dispositivos durante la conducción.
4. **Adelantamientos o cambios de sentido imprudentes**: Realizar maniobras de adelantamiento o cambios de sentido en condiciones de riesgo.
5. **Conducción negligente**: Conducir realizando maniobras imprudentes como acelerones/frenazos, uso inadecuado de carriles o señales de tráfico.
6. **Exceso de velocidad**: Circular sobrepasando el límite de velocidad de la vía o del tramo.
7. **Conducción temeraria**: Conducir realizando maniobras extremadamente peligrosas.
8. **Vehículos de movilidad personal (VMP)**: Uso incorrecto de patinetes eléctricos u otros VMP.
9. **Emisión de gases contaminantes o falta de combustible**: Circular con vehículos contaminantes fuera de normativa o quedarse sin combustible durante la conducción.
10. **Conducción bajo los efectos del alcohol o las drogas**: Circular superando los límites legales de alcohol o con presencia de sustancias estupefacientes.
11. **Abandono del vehículo sin seguridad**: No tomar las precauciones necesarias al estacionar o parar el vehículo.
12. **Actitudes incívicas**: Tener comportamientos inadecuados como molestar al resto de usuarios, tirar objetos en la carretera o no facilitar los datos en caso de accidente.
13. **Motocicletas, ciclomotores y bicicletas**: Circular sin casco, con pasajeros indebidos, en condiciones inadecuadas o estacionamiento prohibido.
14. **Falta de documentación**: Conducir sin tener en regla la documentación del vehículo o la licencia de conducción.
15. **Transporte de mercancías o pasajeros prohibidos**: Incumplimientos en el transporte de materiales, cargas o pasajeros.
16. **Uso incorrecto de los dispositivos de seguridad**: Utilizar de manera inadecuada o no usar el cinturón de seguridad, alumbrado, claxon, distintivos...
17. **Invasión de zona peatonal**: Circular sin respetar la seguridad de peatones o estacionar entorpeciendo el paso.
18. **Acceso a zona restringida**: Acceder sin autorización a zonas de bajas emisiones (ZBE) o áreas de uso restringido a ciertos vehículos.

Y los clústeres generados por DBSCAN se adaptarían a ellos de la siguiente manera:

CLUSTERS DEFINITIVOS		
Clusters DBSCAN	**Cluster**	**Tipo multa**
1, 2, 3, 4, 9, 16, 18, 23, 24 y 26	1	estacionamiento indebido
19, 28, 31, 32, 34, 39, 43, 48, 49 y 50	2	desobediencia a agentes
20	3	uso indebido de dispositivos electrónicos
22 y 30	4	adelantamientos o cambios de sentido imprudentes
5, 7, 8, 14, 21, 36, 38, 40, 42, 44, 46 y 51	5	conducción negligente
3, 6 y 47	6	exceso de velocidad
27 y 45	7	conducción temeraria
29	8	vehículos de movilidad personal (VMP)
33 y 52	9	emisión de gases contaminantes o falta de combustible
10	10	conducción bajo los efectos del alcohol o las drogas
11 y 53	11	abandono del vehículo sin seguridad
12 y 25	12	actitudes incívicas
13 y 37	13	motocicletas, ciclomotores y bicicletas
35 y 47	14	falta de documentación
15	15	transporte de mercancías o pasajeros prohibidos
	16	uso incorrecto de los dispositivos de seguridad
17 y 41	17	invasión de zona peatonal
0	18	acceso a zona restringida

Ilustración 29: Clústeres de 'hecho' definitivos

En base a la tabla, se gestionó a través de Python el mapeo de clústeres y se asignaron los nuevos valores correspondientes al campo 'cluster'. Después, se reasignaron los números de clúster a los valores de 'hecho' revisados manualmente que deberían corresponder a un clúster distinto, que en total fueron 59. Por último, se mapearon las categorías de los clústeres definitivos y se creó una nueva columna llamada 'tipo_multa' con el tipo de multa que corresponde a cada clúster.

Aparte de esto, se eliminó la columna 'embedding', que se había creado automáticamente con el procesamiento, y también la columna 'frecuencia', que contenía la suma de valores de 'hecho' para cada clúster y que se creó con el fin de apoyar el análisis, pero ya no era necesaria.

Tras revisar los resultados en Power BI se verificó que había quedado todo como debería.

CLUSTERS DEFINITIVOS

cluster	tipo_multa	recuento
1	estacionamiento indebido	83
2	desobediencia a agentes	116
3	uso indebido de dispositivos electrónicos	8
4	adelantamientos o cambios de sentido imprudentes	12
5	conducción negligente	143
6	exceso de velocidad	16
7	conducción temeraria	52
8	vehículos de movilidad personal (VMP)	8
9	emisión de gases contaminantes	18
10	conducción bajo efectos del alcohol o drogas	6
11	abandono del vehículo sin seguridad	14
12	actitudes incívicas	18
13	motocicletas, ciclomotores y bicicletas	24
14	falta de documentación	12
15	transporte de mercancías o pasajeros	12
16	uso incorrecto de dispositivos de seguridad	9
17	invasión de zona peatonal	29
18	acceso a zona restringida	5
Total		585

hecho
distintivo no valido: uso de tepmr en zona ser verde
estacionar camion de mas de 12 tn o autobus de mas de 7 mts er
estacionar con autorizacion en lugar habilitado para el estacionan
estacionar con autorizacion no valida
estacionar con autorizacion no valida.
estacionar con distintivo no valido.
estacionar con un titulo que no habilita para la reserva de carga y
estacionar dificultandoel giro o impidiendo el paso de vehiculos.
estacionar el vehiculo en via publica para el ejercicio de una activi
estacionar en autopista o autovia, en zona no habilitada para ello
estacionar en bateria, sin senalizacion que lo autorice expresamer
estacionar en carril bici obstaculizando la circulacion o constituye
estacionar en carril bus.
estacionar en carril de circulacion.
estacionar en carril decirculacion obstaculizando gravemente la ci
estacionar en carril reservado para uso, parada o estacionamiento
estacionar en curva o en cambio de rasante con visibilidad insufic
estacionar en doble fila.
estacionar en el arcen obstaculizando la circulacion o constituyen

Ilustración 30: Ejemplo de los resultados de la clusterización de 'hecho'

Por último, se anexaron las columnas 'cluster' y 'tipo_multa' al dataset de multas de circulación. Para ello se realizó un "left join" que mantuviese todas las filas del dataset e incluyese las dos columnas del CSV de clústeres a través de la coincidencia de la columna común en ambos ficheros: 'hecho'.

```
# Realizar un join en base a la columna 'hecho'
multas_con_clusters = multas.merge(clusters, on='hecho', how='left')
```

Código 11: "Left join" para añadir 'cluster' y 'tipo_multa' al dataset

```
Información de multas_10_final:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 527022 entries, 0 to 527021
Data columns (total 13 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   calificacion  527022 non-null  object
 1   lugar         527022 non-null  object
 2   mes           527022 non-null  int64
 3   anio          527022 non-null  int64
 4   importe       527022 non-null  int64
 5   puntos        527022 non-null  int64
 6   denunciante   527022 non-null  object
 7   hecho         527022 non-null  object
 8   vel_limite    527022 non-null  int64
 9   vel_exceso    527022 non-null  int64
 10  via           527022 non-null  object
 11  cluster       527022 non-null  int64
 12  tipo_multa    527022 non-null  object
```

Ilustración 31: Estructura final del dataset

En el futuro, cuando se quieran introducir nuevos datos de entrenamiento al modelo, se podría crear un modelo supervisado de clasificación con NLP a través de este

clustering que se ha confeccionado, para predecir así a qué clúster corresponde cada nuevo valor de 'hecho', y no tener que volver a realizar todo este proceso de clusterización, revisión y asignación manual de categorías, que ha sido muy costoso.

3.6.3. Resultados

Tras los procesamientos de Fuzzy Matching y Clustering para obtener las columnas 'via', 'cluster' y 'tipo_multa', se disponía de un dataset simplificado y bien estructurado, listo para múltiples tipos de análisis.

Ya que en el análisis exploratorio no se pudieron analizar estos campos debido a su complejidad, se decidió realizar en este punto un pequeño análisis de estos datos.

Primero, se creó una gráfica de barras con matplotlib para obtener las vías Top 20 en volumen de multas graves/muy graves:

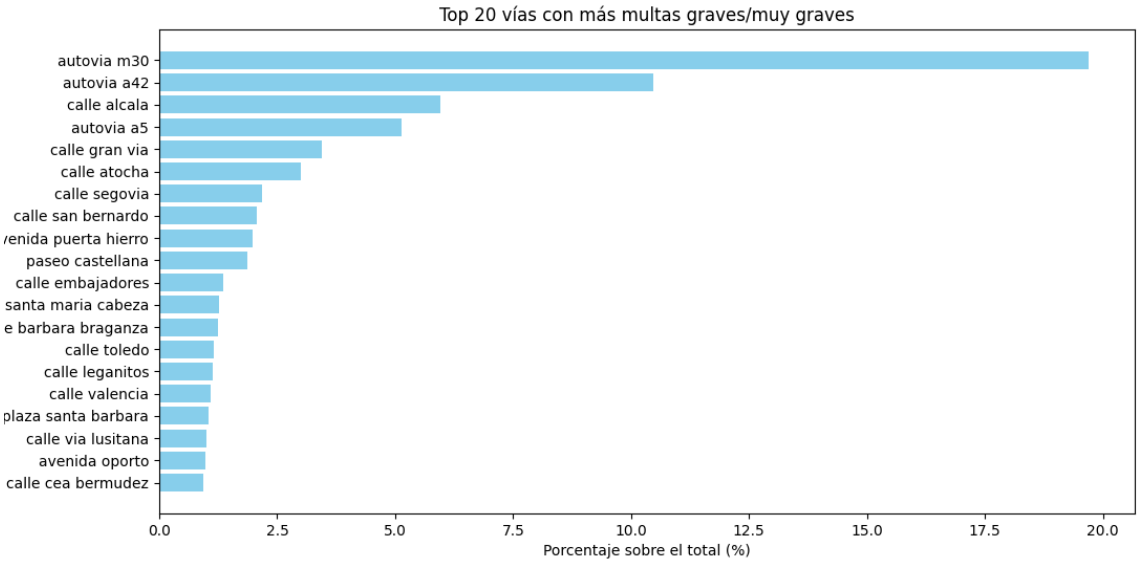


Ilustración 32: Top 20 de vías con más multas graves/muy graves

Las autovías M-30, A-42 y A-5, junto con calle Alcalá, son las vías con mayor volumen de multas graves/muy graves.

Después se confeccionó un mapa de calor o heatmap con Seaborn para visualizar fácilmente de qué tipos son las multas graves/muy graves del mismo Top 20:

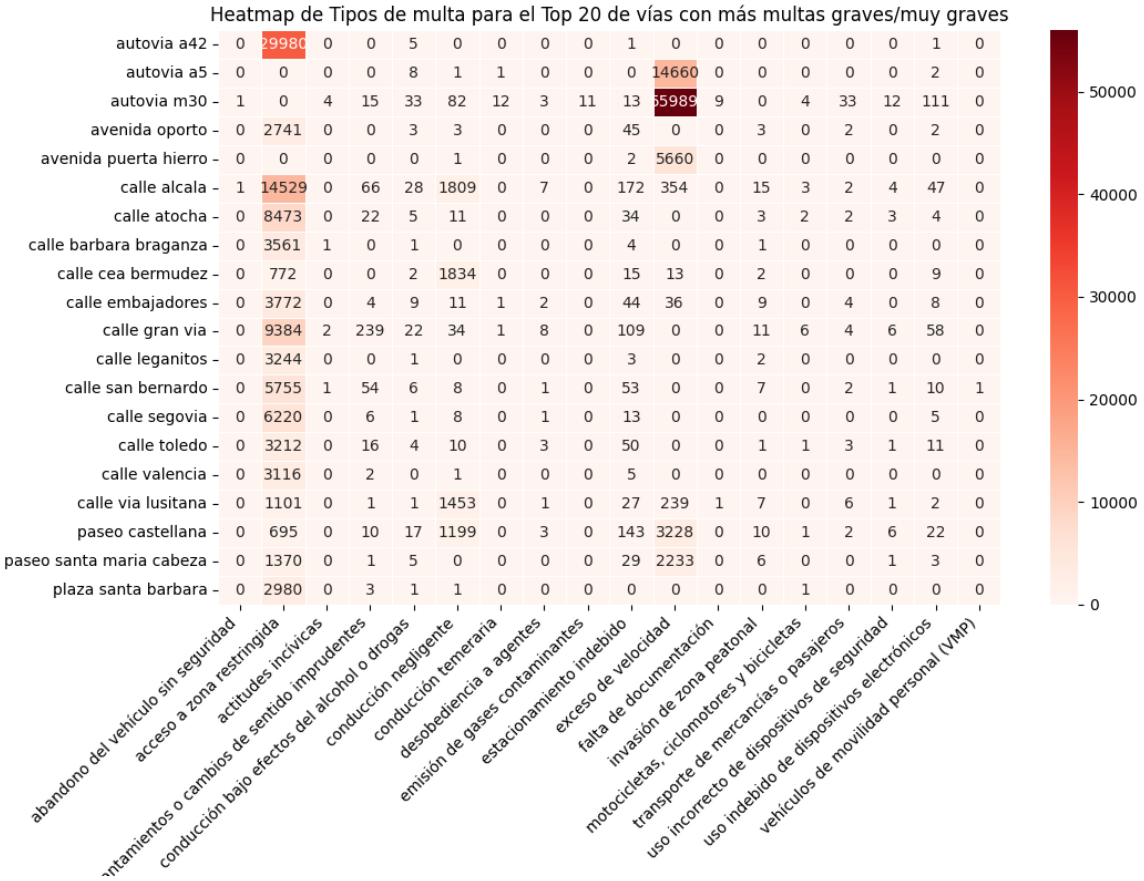


Ilustración 33: Heatmap de 'tipo_multa' para el top 20 de vías con multas graves/muy graves

Aquí se observa que hay un gran volumen de multas "acceso a zona restringida" para autovía A-42 o calle Alcalá y de multas "exceso de velocidad" para autovía A-5 o autovía M-30.

A continuación, se creó un boxplot para comprobar si la vía influye en el importe de la multa:

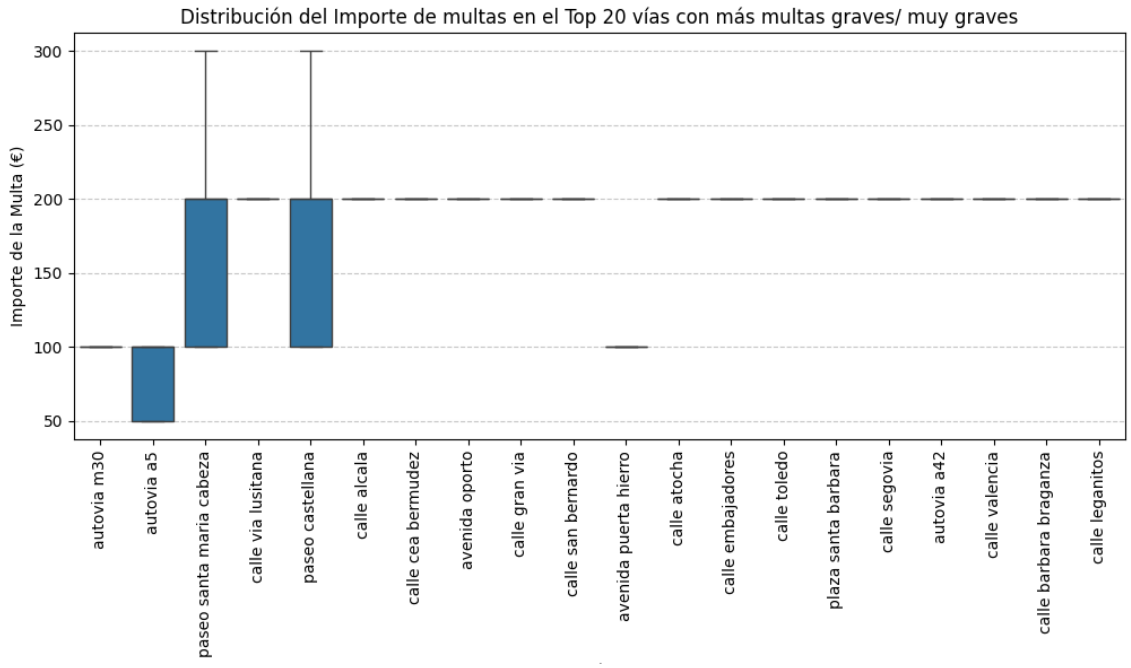


Ilustración 34: Distribución de 'importe' para el top 20 de vías con multas graves/muy graves

Se observa que, así como la mayoría de las vías solo tienen multas graves de un solo importe, hay tres en las que el importe varía considerablemente, por lo tanto, se puede decir que la vía sí influye en el importe de la multa, aunque evidentemente la relación está estrechamente vinculada al hecho causante, como ocurría por ejemplo con el exceso de velocidad, que no es el mismo importe de multa el hecho de exceder la velocidad permitida 40 km/h en una autovía que en una calle urbana.

Lo mismo ocurre con los puntos:

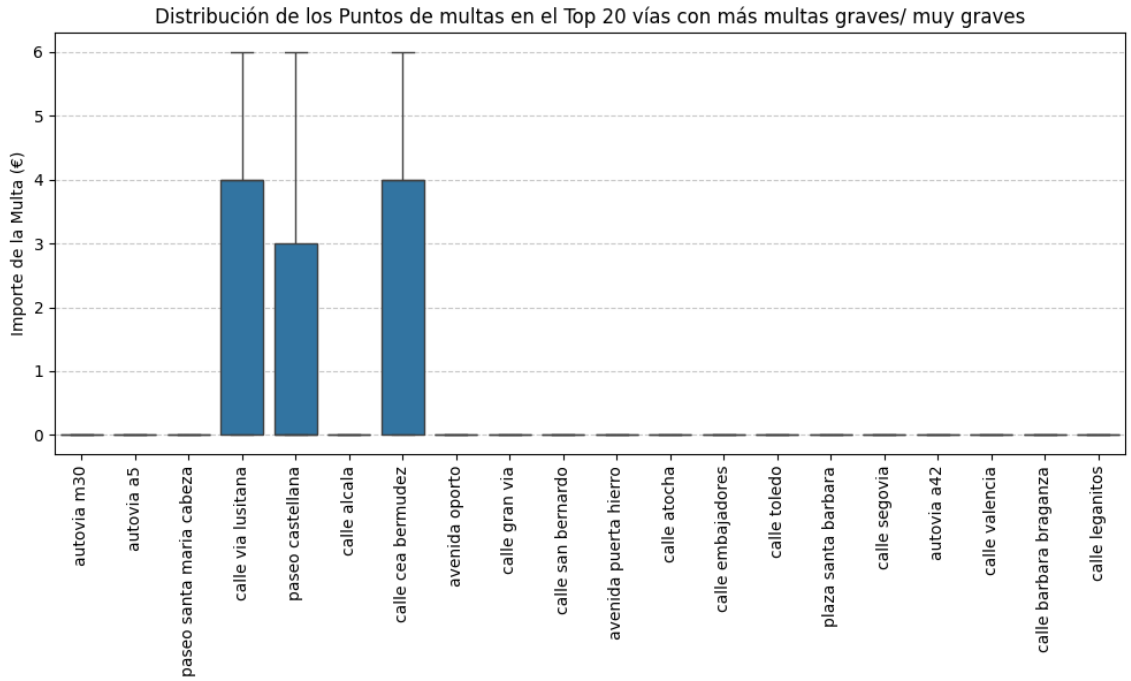


Ilustración 35: Distribución de 'puntos' para el top 20 de vías con multas graves/muy graves

En cuanto al tipo de multa para multas graves y muy graves, destacan 2 categorías, “acceso a zona restringida” y “exceso de velocidad”:

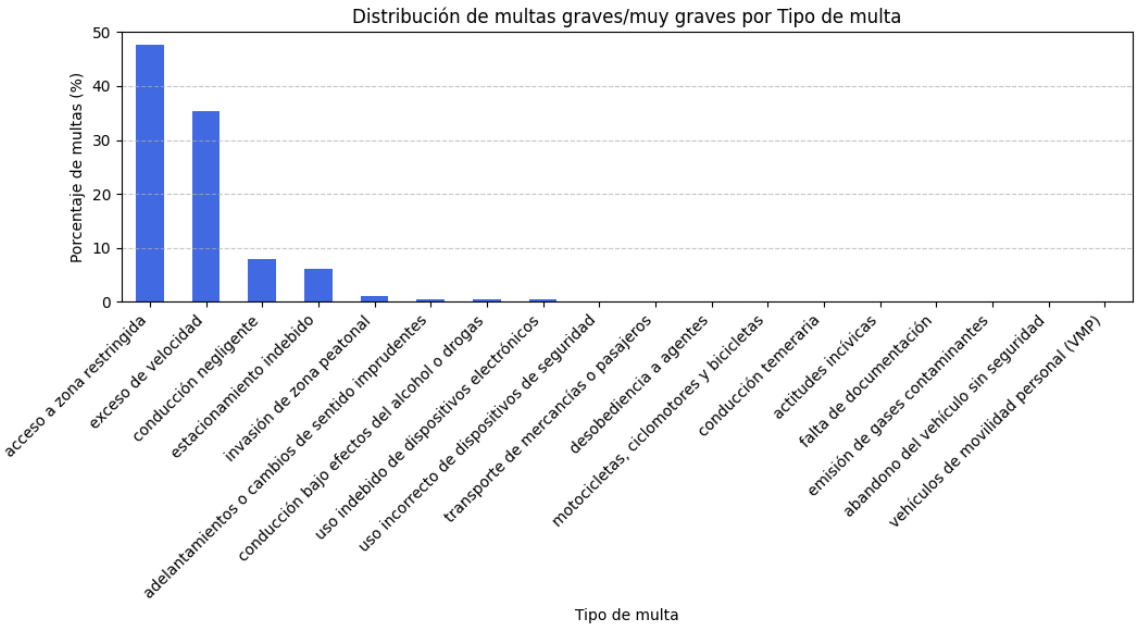


Ilustración 36: Distribución de multas graves/muy graves por ‘tipo_multa’

Para comprobar la relación entre el tipo de multa y el importe, se utilizó de nuevo un boxplot:

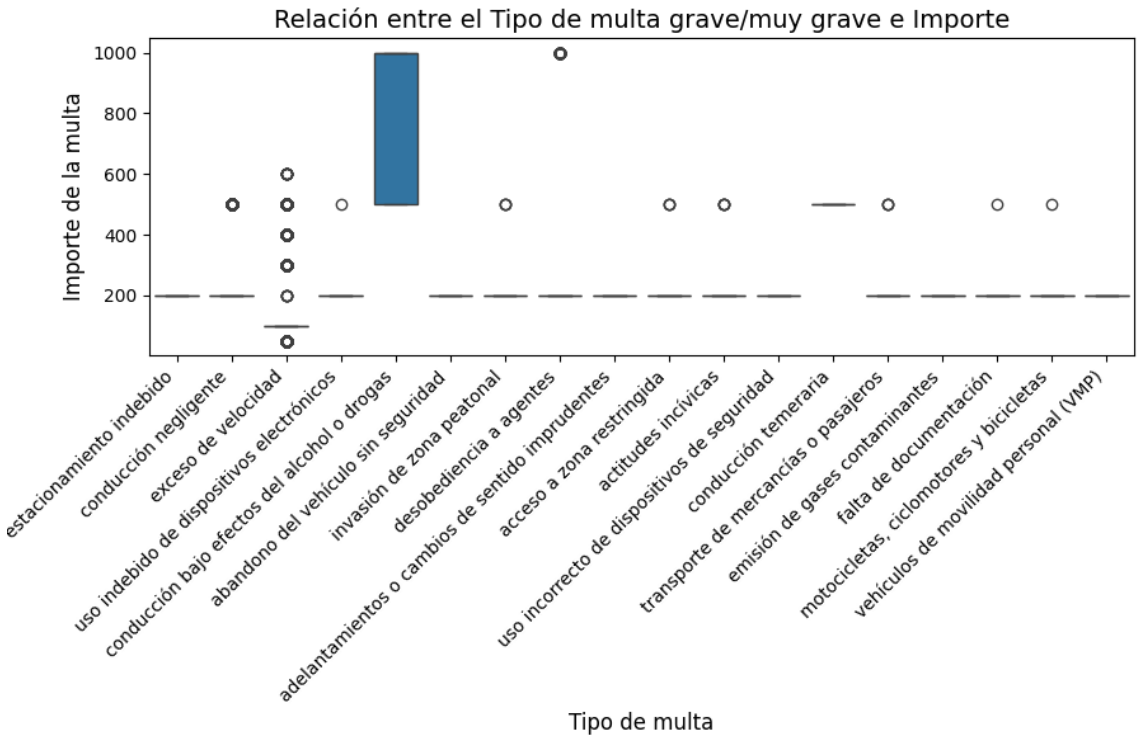


Ilustración 37: Relación entre 'tipo_multa' e 'importe' (multas graves/muy graves)

En la mayoría predominan un único importe con outliers de otros importes. En el caso del tipo "conducción bajo efectos del alcohol o drogas" la variabilidad es más amplia, y lo más probable es que influya la tasa de alcohol arrojada o si acumula efectos de otras drogas. Queda muy claro que el tipo de multa es determinante en el importe de esta, y que los outliers son el resultado de la influencia de otras variables, como se ha podido observar en el resto del análisis, además de que puedan existir otros factores no disponibles como la acumulación de varias infracciones en la misma multa.

3.7. Creación de un modelo de Machine Learning

3.7.1. Consideraciones previas

El objetivo del modelo de Machine Learning creado es predecir la cantidad y el tipo de multas graves y muy graves que se producirán en cada vía en un mes determinado.

Llegados a este punto, se disponía de un dataset con una multa por fila y las siguientes columnas:

- **calificacion**: leve, grave, muy grave.
- **lugar**: a partir de este campo se ha creado la columna 'via', por lo que ya no es necesario para el modelo de ML.
- **mes**: categorías del 1 al 12 que representan el mes correspondiente.
- **anio**: 2022, 2023 y 2024.

- **importe**: 11 categorías de importe (números enteros).
- **puntos**: 6 categorías de puntos (números enteros).
- **denunciante**: 6 categorías (texto).
- **hecho**: a partir de este campo se crearon las columnas ‘cluster’ y ‘tipo_multa’, por lo que ya no es necesario para el modelo de ML.
- **vel_limite**: 9 categorías diferentes (números enteros).
- **vel_exceso**: se creó esta columna a través de la diferencia entre los valores de ‘vel_circula’ (ya eliminada) y ‘vel_exceso’. Son números enteros.
- **via**: resultado del Fuzzy Matching de ‘lugar’ (texto).
- **cluster**: resultado del proceso de clusterización de la columna ‘hecho’. Números enteros del 1 al 18.
- **tipo_multa**: resultado del análisis de los clústeres obtenidos en el proceso de clusterización. Resume los motivos de las infracciones que engloba cada clúster (18 categorías tipo texto). Esta columna no es necesaria para crear el modelo de ML, pero sí posteriormente para interpretar los resultados obtenidos.

Variable objetivo: La variable objetivo es la cantidad de multas calificadas como grave y muy grave, no de manera independiente sino englobando ambas (el volumen de multas muy graves es muy pequeño), para cada cluster.

Problema: El tipo de problema es una tarea de regresión multi-output para predecir el número de multas para cada cluster.

3.7.2. Preparación del dataset

Dataset: Se utilizaron los datos de los años 2022, 2023 y los 4 primeros meses de 2024 como dataset de entrenamiento y el mes de mayo de 2024 como test. El mes de junio se reservó para la evaluación final.

Métrica de evaluación: Como métrica de evaluación, al ser un modelo de regresión, se decidió usar el RMSE.

Dado que el problema a resolver implica temporalidad, pero los modelos de regresión no capturan series temporales como tal, fue necesario añadir agregaciones para ayudar al modelo a capturar las diferencias en el tiempo de las features disponibles.

Con este fin, antes de comenzar con la construcción del modelo, se realizaron las siguientes modificaciones en Python para simplificar el preprocesamiento:

- Eliminación de las columnas ‘lugar’ y ‘hecho’.
- Transformación del campo ‘calificacion’ al tipo binario: “leve” 0 / “grave” y “muy grave” 1.
- Filtrado del dataset por ‘calificacion’ igual a 1 para obtener solo las multas graves y muy graves.
- Creación de columna ‘fecha’ a partir de ‘mes’ y ‘anio’ con formato datetime “AAAA-MM-01”.
- Creación de una nueva columna llamada ‘clase_via’ a partir de la primera palabra de los valores de ‘via’: calle, camino, autovía, etc., para enriquecer las features.

- Creación de una nueva columna llamada ‘estacion’ con las estaciones del año en función de ‘mes’, puesto que podría ser útil para que el modelo aprenda a captar temporalidades, ya que en el análisis se observó que las multas aumentaban o disminuían en ciertos periodos.
    - Mes 12, 1 y 2: Invierno
    - Mes 3, 4 y 5: Primavera
    - Mes 6, 7 y 8: Verano
    - Mes 9, 10 y 11: Otoño
- Agrupación del dataset por ‘via’ y ‘fecha’, de manera que cada fila ya no fuese una multa sino una vía y mes. En consecuencia, se transforman el resto de los campos:
    - Transformación de ‘denunciante’ a 6 campos con el porcentaje de distribución de cada una de sus categorías en el periodo de los últimos 3 meses, sin contar el actual.
    - Transformación de ‘cluster’ a 18 campos con la suma de las multas para cada una de sus categorías. Estas 18 columnas son las variables objetivo.
    - ‘mes’, ‘anio’, ‘estacion’ y ‘clase_via’ se obtienen del primer valor para cada grupo (al estar agrupado por ‘fecha’ y ‘via’ solo hay un valor posible).
    - A partir de ‘importe’, ‘puntos’, ‘vel_limite’ y ‘vel_exceso’ se obtuvieron ‘importe_3m’, ‘puntos_3m’, ‘vel_limite_3m’ y ‘vel_exceso_3m’ respectivamente, que contienen el promedio de valores de los últimos 3 meses, sin contar el actual.
    - Transformación de ‘calificacion’ a ‘multas’, con la suma de sus valores para cada grupo. No se usó en el modelo, solo en el análisis posterior.
    - Nueva columna ‘multas_3m’ con la suma de multas de los últimos 3 meses sin contar el actual.
    - Nuevas columnas ‘multas_6m’ y ‘multas_12m’ con el promedio de multas de los últimos 6 y 12 meses respectivamente, sin contar el actual.
- Conversión de valores vacíos a ceros.

```python
# Agrupar por 'fecha' y 'via' y aplicar las agregaciones
columnas_agregaciones = {
    'mes': 'first',         # Primer valor de 'mes'
    'anio': 'first',        # Primer valor de 'anio'
    'estacion': 'first',    # Primer valor de 'estacion'
    'clase_via': 'first',   # Primer valor de 'clase_via'
    'importe': 'mean',      # Promedio de 'importe'
    'puntos': 'mean',       # Promedio de 'puntos'
    'vel_limite': 'mean',   # Promedio de 'vel_limite'
    'vel_exceso': 'mean',   # Promedio de 'vel_exceso'
}

# Añadir la columna 'cluster' a las agregaciones
for i in range(1, 19):
    columnas_agregaciones[f'cluster_{i}'] = 'sum'  # Suma de cada categoría

# Agrupar y aplicar las agregaciones
dataset_agrupado = dataset_filtrado.groupby(
    ['fecha', 'via']).agg(columnas_agregaciones).reset_index()

# Crear la columna 'multas' como el tamaño de cada grupo
dataset_agrupado['multas'] = dataset_filtrado.groupby(
    ['fecha', 'via']).size().values

# Calcular promedios/sumas móviles de los últimos 3 meses (exc mes actual)
def calcular_promedio_3m(df, columna, nueva_columna):
    df = df.sort_values(by='fecha')
    # Para 'multas' se usa suma, para las demás el promedio
    if columna == 'multas':
        df[nueva_columna] = df[columna].shift(1) \
            .rolling(window=3, min_periods=1).sum()
    else:
        df[nueva_columna] = df[columna].shift(1) \
            .rolling(window=3, min_periods=1).mean()
    return df

columnas_promedio_3m = ['importe', 'puntos', 'vel_limite',
                        'vel_exceso', 'multas']
for columna in columnas_promedio_3m:
    dataset_agrupado = dataset_agrupado.groupby('via').apply(
        lambda x: calcular_promedio_3m(x, columna, f'{columna}_3m')
    ).reset_index(drop=True)
```

Código 12: Agrupación de los datos por 'via' y 'fecha' (parte I)

```python
# Calcular % de las categorías de 'denunciante' en los últimos 3 meses
def calcular_porcentajes_denunciante(df, categoria):
    df = df.sort_values(by='fecha')
    df[f'denunciante_{categoria}_3m'] = df[f'denunciante_{categoria}'] \
        .shift(1).rolling(window=3, min_periods=1).mean() * 100
    return df

for categoria in denunciantes_categorias:
    dataset_agrupado = dataset_agrupado.groupby('via').apply(
        lambda x: calcular_porcentajes_denunciante(x, categoria)
    ).reset_index(drop=True)

# Calcular columnas con promedio de multas de últimos 6 y 12m (exc mes actual)
def calcular_promedio_multas(df, window, nueva_columna):
    df = df.sort_values(by='fecha')
    df[nueva_columna] = df['multas'].shift(1) \
        .rolling(window=window, min_periods=1).mean()
    return df

dataset_agrupado = dataset_agrupado.groupby('via').apply(
    lambda x: calcular_promedio_multas(x, 6, 'multas_6m')
).reset_index(drop=True)

dataset_agrupado = dataset_agrupado.groupby('via').apply(
    lambda x: calcular_promedio_multas(x, 12, 'multas_12m')
).reset_index(drop=True)
```

Código 13: Agrupación de los datos por 'via' y 'fecha' (parte II)

3.7.3. Preprocesamiento

Variable objetivo: Como se comenta en el apartado anterior, las variables objetivo son la cantidad de multas para cada uno de los 18 clústeres. Estos campos estuvieron presentes en el entrenamiento, tanto en train como en test, para poder evaluar el rendimiento del modelo. Sin embargo, no estuvieron presentes en el dataset de evaluación final (junio de 2024) ya que son los datos para predecir y, en un caso de uso real donde queramos predecir el futuro, no dispondremos de estos datos.

Features: En cuanto a las features del modelo, finalmente fueron:

- fecha
- clase_via
- estacion
- denunciante_policia municipal_3m
- denunciante_movilidad radar_3m
- denunciante_medios de captacion de imagen_3m
- denunciante_agentes de movilidad_3m
- denunciante_sace_3m
- denunciante_ser_3m
- importe_3m, puntos_3m
- vel_limite_3m
- vel_exceso_3m
- multas_3m

- multas_6m
- multas_12m

La feature ‘fecha’ no puede ser procesada por el modelo como tipo datetime, por lo que durante el preprocesamiento fue convertida a tipo numérico con timestamp (segundos desde época Unix).

```
# Convertir 'fecha' a formato datestamp (timestamp Unix en segundos)
df['fecha'] = df['fecha'].astype(np.int64) // 10**9
```

Código 14: Convertir ‘fecha’ a datestamp

Solo hay 2 variables categóricas entre las features, ‘clase_via’ y ‘estacion’. También fue necesario codificarlas como numéricas. En este caso se preprocesaron con OneHotEncoder, que convierte cada categoría del campo en una columna binaria. Solo se configuraron dos parámetros:

- **sparse_output**: Controla si la salida es una matriz dispersa o una densa.
  - True (predeterminado): Matriz dispersa, eficiente en memoria.
  - False: Matriz densa, útil si se necesita manipular datos directamente, como es el caso.
- **handle_unknown**: Indica qué hacer con categorías desconocidas en transform.
  - “error” (predeterminado): Devuelve error.
  - “ignore”: Las ignora y codifica ceros. Se decide usar esta opción.

```
# Codificar variables categóricas utilizando OneHotEncoder
categorical_features = ['estacion', 'clase_via']
onehot_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Ajustar el encoder y transformar los datos
encoded_train = onehot_encoder.fit_transform(X_train[categorical_features])
encoded_test = onehot_encoder.transform(X_test[categorical_features])
```

Código 15: Codificación de variables categóricas con OneHotEncoder

El resto de features son numéricas, se podrían procesar tal cual están, pero como las escalas son muy diferentes entre unas y otras se utilizó como técnica de normalización la estandarización (StandardScaler), que no es más que restar la media y dividir por la desviación estándar.

```python
# Normalizar variables numéricas
scaler = StandardScaler()
num_features = ['multas_3m', 'importe_3m', 'puntos_3m', 'vel_limite_3m',
                'vel_exceso_3m', 'multas_6m', 'multas_12m']
X_train[num_features] = scaler.fit_transform(X_train[num_features])
X_test[num_features] = scaler.transform(X_test[num_features])
```

Código 16: Normalización de variables numéricas con StandarScaler

3.7.4. Entrenamiento del modelo

Para el entrenamiento se utilizaron cuatro modelos diferentes:

- Random Forest
- CatBoost
- XGBoost
- LightGBM

Se eligieron estos modelos porque los cuatro son buenas opciones para el problema de regresión, además de que son muy eficientes para manejar relaciones no lineales, interacciones entre características y estructuras de datos tabulares.

Por otro lado, se optó por cuatro modelos para poder comparar resultados entre ellos y elegir el de mayor rendimiento. Para este fin, el primer paso fue realizar búsquedas de los mejores hiperparámetros para cada uno de ellos.

Además, en el entrenamiento fue necesario implementar un MultiOutputRegressor para aplicar una regresión independiente a cada uno de los objetivos (targets). Para ello, se construyó una clase personalizada:

```python
# Clase personalizada Custom multioutput regressor
# que permite entrenar un modelo XGBoost para cada target
class CustomMultiOutputRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, estimator):
        self.estimator = estimator

    def fit(self, X, Y, **fit_params):
        self.estimators_ = []
        for i in range(Y.shape[1]):
            # Extraer la i-ésima columna del target
            y = Y.iloc[:, i] if hasattr(Y, "iloc") else Y[:, i]
            estimator = clone(self.estimator)
            estimator.fit(X, y, **fit_params)
            self.estimators_.append(estimator)
        return self

    def predict(self, X):
        preds = [estimator.predict(X) for estimator in self.estimators_]
        return np.column_stack(preds)
```

Código 17: Clase personalizada para MultiOutputRegressor

- **BaseEstimator**: Permite que la clase sea compatible con el ecosistema de Scikit-learn.
- **RegressorMixin**: Indica que esta clase representa un modelo de regresión.
- **Función fit**: Entrena cada modelo desde cero y lo guarda en la lista self.estimators_.
- **Función predict**: Para cada modelo almacenado en self.estimators_calcula una predicción, las combina y las almacena en una matriz 2D.

3.7.5. Ajuste de hiperparámetros

La librería que se usó para la búsqueda de hiperparámetros fue de nuevo Optuna. Además, se implementó validación cruzada temporal (TimeSeriesSplit) en los cuatro modelos y también Early Stopping en los tres modelos de boosting, ya que Random Forest no lo admite.

- Modelo XGboost

Existen múltiples hiperparámetros que se pueden configurar, pero en este caso se optó por los siguientes:

- **learning_rate**: determina la velocidad a la que el algoritmo de boosting aprende de cada iteración. Un valor más bajo significa un aprendizaje más lento, ya que reduce la contribución de cada árbol del conjunto.
- **max_depth**: representa la profundidad a la que puede crecer cada árbol del proceso de boosting durante el entrenamiento.
- **n_estimators**: especifica el número de árboles que se construirán en el conjunto.
- **subsample**: Proporción de muestreo de las instancias de entrenamiento.
- colsample_bytree: Es la proporción de muestreo de las columnas al construir cada árbol. El muestreo se realiza una vez por cada árbol construido.
- **gamma**: controla la cantidad mínima de reducción de pérdidas necesaria para realizar una nueva división en un nodo hoja del árbol.
- **min_child_weight**: Suma mínima del peso de las instancias (hessiana) requerida en un nodo hijo.
- **reg_alpha**: Término de regularización L1 en los pesos. Aumentar este valor hará que el modelo sea más conservador.
- **reg_lambda**: Término de regularización L2 en los pesos. Aumentar este valor hará que el modelo sea más conservador.

```python
def objective(trial):
    # Definir rangos de hiperparámetros para XGBoost
    params = {
        'objective': 'reg:squarederror',
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.2),
        'max_depth': trial.suggest_int('max_depth', 3, 8),
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'subsample': trial.suggest_uniform('subsample', 0.5, 0.9),
        'colsample_bytree': trial.suggest_uniform('colsample_bytree', 0.5, 0.9),
        'gamma': trial.suggest_loguniform('gamma', 1e-8, 5.0),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'reg_alpha': trial.suggest_loguniform('reg_alpha', 1e-8, 1.0),
        'reg_lambda': trial.suggest_loguniform('reg_lambda', 1.0, 5.0),
        'verbosity': 0,
        'random_state': 42,
        'early_stopping_rounds': 10
    }
```

Código 18: Optimización de hiperparámetros con Optuna para XGBoost

Se realizaron varias búsquedas estableciendo primero rangos amplios de hiperparámetros para irlos acotando después, además de probar varios números de splits distintos para TimeSeriesSplit, cuyo mejor resultado fue 6, y diferentes rounds para Early Stopping, cuyo mejor resultado fue 10.

```
====================================
Mejores hiperparámetros encontrados:
learning_rate: 0.1624412945030855
max_depth: 4
n_estimators: 243
subsample: 0.6948992876645413
colsample_bytree: 0.506372477225035
gamma: 0.6627777063062757
min_child_weight: 1
reg_alpha: 1.533216193611394e-05
reg_lambda: 2.380852438377045
====================================

Resultados de RMSE en test:
Cluster_1 RMSE: 1.0249
Cluster_2 RMSE: 0.1021
Cluster_3 RMSE: 0.3043
Cluster_4 RMSE: 0.4226
Cluster_5 RMSE: 2.4592
Cluster_6 RMSE: 13.9229
Cluster_7 RMSE: 0.0786
Cluster_8 RMSE: 0.0002
Cluster_9 RMSE: 0.0002
Cluster_10 RMSE: 0.2985
Cluster_11 RMSE: 0.0006
Cluster_12 RMSE: 0.0699
Cluster_13 RMSE: 0.0782
Cluster_14 RMSE: 0.0495
Cluster_15 RMSE: 0.0994
Cluster_16 RMSE: 0.1204
Cluster_17 RMSE: 0.4495
Cluster_18 RMSE: 21.1098
====================================
```

Ilustración 38: Resultados de la búsqueda de hiperparámetros para XGBoost

Tras anotar los resultados de cada tanda realizada, se usó un modelo base de XGBoost, sin Optuna, TimeSeriesSplit ni Early Stopping, para realizar unas últimas búsquedas manuales y afinar al máximo posible los resultados de RMSE.

```python
# Definir hiperparámetros fijos para XGBoost
params = {
    'objective': 'reg:squarederror',
    'learning_rate': 0.01362984193504081,
    'max_depth': 8,
    'n_estimators': 600,
    'subsample': 0.9994296262006694,
    'colsample_bytree': 0.6015260164418761,
    'gamma': 4,
    'min_child_weight': 4,
    'reg_alpha': 1,
    'reg_lambda': 4.7709642977192024e-05,
    'verbosity': 0,
    'random_state': 42
}

# Entrenar el modelo utilizando el custom multioutput regressor
final_model = CustomMultiOutputRegressor(xgb.XGBRegressor(**params))
final_model.fit(X_train, y_train)

# Predicciones y evaluación
y_pred = final_model.predict(X_test)
final_rmse_scores = [
    np.sqrt(mean_squared_error(y_test.iloc[:, i], y_pred[:, i]))
    for i in range(len(targets))
]

# Imprimir hiperparámetros utilizados en el modelo
print("====================================")
print("Hiperparámetros utilizados en el modelo:")
for key, value in params.items():
    print(f"{key}: {value}")
print("====================================")

# Imprimir resultados de RMSE para cada cluster
print("Resultados de RMSE en test:")
for i, score in enumerate(final_rmse_scores, 1):
    print(f'Cluster_{i} RMSE: {score:.4f}')
print("====================================")
```

Código 19: Modelo base XGBoost para testear predicciones

Finalmente, los mejores resultados obtenidos fueron:

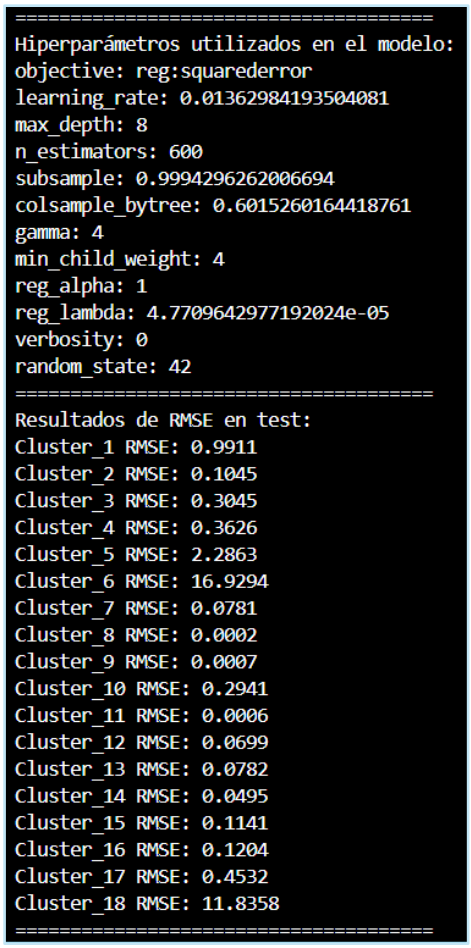


Ilustración 39: Mejores hiperparámetros encontrados para XGBoost

- Modelo CatBoost

Los hiperparámetros que se configuraron para este modelo fueron:

- **loss_function**: Métricas de evaluación de los datos de validación.
- **learning_rate**: La velocidad a la que se actualizan los pesos del modelo después de pasar por cada lote de ejemplos de entrenamiento.
- **depth**: Profundidad del árbol.
- **iterations**: La cantidad máxima de árboles que se puede construir.
- **l2_leaf_reg**: Coeficiente del término de regularización L2 de la función de coste.
- **bagging_temperature**: Define la configuración del arranque bayesiano.
- **random_strength**: La cantidad de asignación al azar que se utilizará para puntuar las divisiones cuando se selecciona la estructura de árbol. Útil para evitar sobreajustar el modelo.
- **od_type**: Define el criterio que se usará para detener el entrenamiento si detecta sobreajuste.
    - "IncToDec": Detiene el entrenamiento cuando la métrica de evaluación en el conjunto de validación ha dejado de mejorar durante od_wait iteraciones consecutivas.
    - "Iter": Simplemente detiene el entrenamiento después de un número fijo de iteraciones. Se optó por esta opción.

- **od_wait**: define cuántas iteraciones debe esperar CatBoost sin ver mejoras en la métrica de evaluación antes de detener el entrenamiento.

```python
def objective(trial):
    params = {
        'loss_function': 'RMSE',
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.1),
        'depth': trial.suggest_int('depth', 3, 6),
        'iterations': trial.suggest_int('iterations', 100, 1000),
        'l2_leaf_reg': trial.suggest_loguniform('l2_leaf_reg', 3, 10),
        'bagging_temperature': trial.suggest_uniform(
            'bagging_temperature', 0, 1),
        'random_strength': trial.suggest_uniform('random_strength', 0, 1),
        'od_type': 'Iter',
        'od_wait': 10,
        'verbose': False,
        'random_seed': 42
    }
```

Código 20: Optimización de hiperparámetros con Optuna para CatBoost

Se realizó exactamente el mismo proceso que en el modelo anterior, obteniendo como mejores resultados los siguientes:

```
====================================
Hiperparámetros utilizados en el modelo:
loss_function: RMSE
learning_rate: 0.006416135040893888
depth: 10
iterations: 1000
l2_leaf_reg: 4.146106423105672e-05
bagging_temperature: 0.9902177259077334
random_strength: 0.8613731086597908
verbose: False
random_seed: 42
====================================
Resultados de RMSE en test:
Cluster_1 RMSE: 0.9866
Cluster_2 RMSE: 0.0993
Cluster_3 RMSE: 0.3187
Cluster_4 RMSE: 0.3568
Cluster_5 RMSE: 1.9142
Cluster_6 RMSE: 16.3720
Cluster_7 RMSE: 0.0839
Cluster_8 RMSE: 0.0026
Cluster_9 RMSE: 0.0048
Cluster_10 RMSE: 0.3181
Cluster_11 RMSE: 0.0310
Cluster_12 RMSE: 0.0738
Cluster_13 RMSE: 0.0796
Cluster_14 RMSE: 0.0495
Cluster_15 RMSE: 0.1026
Cluster_16 RMSE: 0.1241
Cluster_17 RMSE: 0.4480
Cluster_18 RMSE: 13.6481
====================================
```

Ilustración 40: Mejores hiperparámetros encontrados para CatBoost

- Modelo Random Forest

Los hiperparámetros que se configuraron para este modelo fueron:

- **n_estimators**: número de árboles incluidos en el modelo.
- **max_depth**: profundidad máxima que pueden alcanzar los árboles.
- **min_samples_split**: número mínimo de observaciones que debe de tener un nodo para que pueda dividirse.
- **min_samples_leaf**: número mínimo de observaciones que debe de tener cada uno de los nodos hijos para que se produzca la división.

```python
def objective(trial):
    # Definir rangos de hiperparámetros para RandomForestRegressor
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 900),
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 10),
        'max_features': trial.suggest_categorical(
            'max_features', ['sqrt', 'log2', None]),
        'random_state': 42,
        'n_jobs': -1
    }
```

Código 21: Optimización de hiperparámetros con Optuna para Random Forest

Siguiendo el mismo procedimiento que en los casos anteriores, salvo la ejecución de Early Stopping que no es aplicable para este modelo, los mejores resultados obtenidos fueron:

```
====================================
Hiperparámetros utilizados en el modelo:
n_estimators: 154
max_depth: 8
min_samples_split: 4
min_samples_leaf: 4
random_state: 42
n_jobs: -1
====================================
Resultados de RMSE en test:
Cluster_1 RMSE: 0.9086
Cluster_2 RMSE: 0.1015
Cluster_3 RMSE: 0.3083
Cluster_4 RMSE: 0.3523
Cluster_5 RMSE: 1.8542
Cluster_6 RMSE: 11.3855
Cluster_7 RMSE: 0.0832
Cluster_8 RMSE: 0.0014
Cluster_9 RMSE: 0.0111
Cluster_10 RMSE: 0.3021
Cluster_11 RMSE: 0.0074
Cluster_12 RMSE: 0.0730
Cluster_13 RMSE: 0.0787
Cluster_14 RMSE: 0.0497
Cluster_15 RMSE: 0.0975
Cluster_16 RMSE: 0.1238
Cluster_17 RMSE: 0.4487
Cluster_18 RMSE: 28.4284
====================================
```

Ilustración 41: Mejores hiperparámetros encontrados para Random Forest

- Modelo LightGBM

Los hiperparámetros que se configuraron para este modelo fueron:

- **boosting_type**: selecciona el tipo de modelo a implementar. Tiene varias opciones:
  - 'gbdt': es el parámetro por defecto, implementa el tradicional Gradient Boosting Decision Tree.
  - 'dart': Dropouts Additive Regression Trees
  - 'goss': Gradient-based One-Side Sampling
  - 'rf': Random Forest
- **num_leaves**: es el número máximo de hojas por árbol (31 por defecto).
- **max_depth**: profundidad máxima de árbol (-1 por defecto, números negativos indican que no tiene límite).
- **learning_rate**: es la tasa de aprendizaje (0.1 por defecto).
- **n_estimators**: número total de árboles (100 por defecto).
- **objective**: 'regression' para problemas de regresión y 'binary' o 'multiclass' para problemas de clasificación (None por defecto).
- **min_child_samples**: Número mínimo de datos en una hoja. Se puede utilizar para manejar el sobreajuste.
- **feature_fraction**: Un subconjunto de características que se seleccionarán en cada iteración (árbol).
- **bagging_fraction**: Un subconjunto de características similar a feature_fraction, pero bagging_fraction selecciona aleatoriamente una parte de los datos sin necesidad de volver a muestrearlos.
- **bagging_freq**: La frecuencia con la que se realiza el bagging. En cada iteración de bagging_freq, LightGBM selecciona aleatoriamente un porcentaje de los datos para usarlos en la siguiente iteración bagging_freq. Este porcentaje lo determina el hiperparámetro bagging_fraction. Si bagging_freq es cero, se desactiva el bagging.
- **lambda_l1**: Regularización L1.
- **lambda_l2**: Regularización L2.
- **min_split_gain**: La ganancia mínima para realizar una división. Se puede utilizar para acelerar el entrenamiento.

```python
def objective(trial):
    params = {
        'objective': 'regression',
        'metric': 'rmse',
        'boosting_type': 'gbdt',
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        "learning_rate": trial.suggest_loguniform("learning_rate", 0.01, 0.2),
        "num_leaves": trial.suggest_int("num_leaves", 20, 60),
        "max_depth": trial.suggest_int("max_depth", 3, 8),
        "min_child_samples": trial.suggest_int("min_child_samples", 10, 50),
        "feature_fraction": trial.suggest_uniform("feature_fraction", 0.6, 1.0),
        "bagging_fraction": trial.suggest_uniform("bagging_fraction", 0.6, 1.0),
        "bagging_freq": trial.suggest_int("bagging_freq", 1, 7),
        "lambda_l1": trial.suggest_uniform("lambda_l1", 0, 5),
        "lambda_l2": trial.suggest_uniform("lambda_l2", 0, 5),
        "min_split_gain": trial.suggest_uniform("min_split_gain", 0, 1),
        "verbose": -1
    }
```

Código 22: Optimización de hiperparámetros con Optuna para LightGBM

Los mejores resultados obtenidos para LightGBM fueron:

```
Hiperparámetros utilizados en el modelo:
n_estimators: 900
learning_rate: 0.08060831074193205
num_leaves: 60
max_depth: 7
min_child_samples: 15
feature_fraction: 0.9989959232359616
bagging_fraction: 0.6824781310337036
bagging_freq: 3
lambda_l1: 1.9874195462913222
lambda_l2: 3.5128792054850 5
min_split_gain: 0.553005056881345
objective: regression
metric: rmse
boosting_type: gbdt
verbose: -1
==================================
Resultados de RMSE en test:
Cluster_1 RMSE: 0.9562
Cluster_2 RMSE: 0.1037
Cluster_3 RMSE: 0.3154
Cluster_4 RMSE: 0.4097
Cluster_5 RMSE: 1.6456
Cluster_6 RMSE: 23.4821
Cluster_7 RMSE: 0.0783
Cluster_8 RMSE: 0.0002
Cluster_9 RMSE: 0.0018
Cluster_10 RMSE: 0.2948
Cluster_11 RMSE: 0.0006
Cluster_12 RMSE: 0.0699
Cluster_13 RMSE: 0.0782
Cluster_14 RMSE: 0.0495
Cluster_15 RMSE: 0.1042
Cluster_16 RMSE: 0.1210
Cluster_17 RMSE: 0.4488
Cluster_18 RMSE: 30.0051
==================================
```

Ilustración 42: Mejores hiperparámetros encontrados para LightGBM

3.7.6. Resultados de las predicciones

Al realizar la comparativa de los cuatro modelos, el que mejor rendimiento tuvo en líneas generales fue XGBoost:

XGBOOST	CATBOOST	RANDOM FOREST	LIGHTGBM
Cluster_1 RMSE: 0.9911	Cluster_1 RMSE: 0.9866	Cluster_1 RMSE: 0.9086	Cluster_1 RMSE: 0.9562
Cluster_2 RMSE: 0.1045	Cluster_2 RMSE: 0.0993	Cluster_2 RMSE: 0.1015	Cluster_2 RMSE: 0.1037
Cluster_3 RMSE: 0.3045	Cluster_3 RMSE: 0.3187	Cluster_3 RMSE: 0.3083	Cluster_3 RMSE: 0.3154
Cluster_4 RMSE: 0.3626	Cluster_4 RMSE: 0.3568	Cluster_4 RMSE: 0.3523	Cluster_4 RMSE: 0.4097
Cluster_5 RMSE: 2.2863	Cluster_5 RMSE: 1.9142	Cluster_5 RMSE: 1.8542	Cluster_5 RMSE: 1.6456
Cluster_6 RMSE: 16.9294	Cluster_6 RMSE: 16.3720	Cluster_6 RMSE: 11.3855	Cluster_6 RMSE: 23.4821
Cluster_7 RMSE: 0.0781	Cluster_7 RMSE: 0.0839	Cluster_7 RMSE: 0.0832	Cluster_7 RMSE: 0.0783
Cluster_8 RMSE: 0.0002	Cluster_8 RMSE: 0.0026	Cluster_8 RMSE: 0.0014	Cluster_8 RMSE: 0.0002
Cluster_9 RMSE: 0.0007	Cluster_9 RMSE: 0.0048	Cluster_9 RMSE: 0.0111	Cluster_9 RMSE: 0.0018
Cluster_10 RMSE: 0.2941	Cluster_10 RMSE: 0.3181	Cluster_10 RMSE: 0.3021	Cluster_10 RMSE: 0.2948
Cluster_11 RMSE: 0.0006	Cluster_11 RMSE: 0.0310	Cluster_11 RMSE: 0.0074	Cluster_11 RMSE: 0.0006
Cluster_12 RMSE: 0.0699	Cluster_12 RMSE: 0.0738	Cluster_12 RMSE: 0.0730	Cluster_12 RMSE: 0.0699
Cluster_13 RMSE: 0.0782	Cluster_13 RMSE: 0.0796	Cluster_13 RMSE: 0.0787	Cluster_13 RMSE: 0.0782
Cluster_14 RMSE: 0.0495	Cluster_14 RMSE: 0.0495	Cluster_14 RMSE: 0.0497	Cluster_14 RMSE: 0.0495
Cluster_15 RMSE: 0.1141	Cluster_15 RMSE: 0.1026	Cluster_15 RMSE: 0.0975	Cluster_15 RMSE: 0.1042
Cluster_16 RMSE: 0.1204	Cluster_16 RMSE: 0.1241	Cluster_16 RMSE: 0.1238	Cluster_16 RMSE: 0.1210
Cluster_17 RMSE: 0.4532	Cluster_17 RMSE: 0.4480	Cluster_17 RMSE: 0.4487	Cluster_17 RMSE: 0.4488
Cluster_18 RMSE: 11.8358	Cluster_18 RMSE: 13.6481	Cluster_18 RMSE: 28.4284	Cluster_18 RMSE: 30.005

Ilustración 43: Comparativa de los resultados de RMSE de los cuatro modelos testeados

Como se puede observar en la imagen, hay clústeres en los que el RMSE es muy bajo y otros, como el 6 o el 18, en los que el RMSE es mayor. Un RMSE alto no siempre significa que el modelo sea peor, sino que puede indicar que en ese clúster hay valores mucho más grandes o mayor variabilidad, lo cual tiene sentido ya que los clústeres 6 y 18 (exceso de velocidad y acceso a zona restringida respectivamente), son los que mayor volumen de multas tienen, como se verificó en el análisis. Ahora bien, entre distintos modelos, será mejor el que tenga menor RMSE para el mismo clúster. Ningún modelo resultó mejor para todos los clústeres, pero XGBoost es el que obtuvo mayor cantidad de clústeres con menor RMSE y ninguno de sus clústeres resultó en un RMSE demasiado alto.

Por lo tanto, se eligió finalmente XGBoost como modelo final y se procedió a analizar las predicciones obtenidas contra los valores reales a través de gráficos de dispersión para cada clúster:

- **Clúster 1**: Estacionamiento indebido (aprox. el 6% de las multas). Se aprecia que para valores bajos la precisión es mejor. Para valores reales más altos, la predicción suele estar por debajo, aunque también hay algunas predicciones por encima.

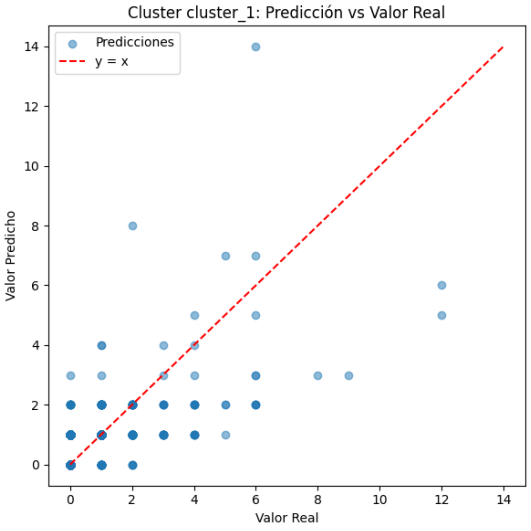


Ilustración 44: Predicción vs Valor real en Clúster 1

- **Clúster 2**: Desobediencia a agentes (menos del 1% de las multas).
- **Clúster 7**: Conducción temeraria (menos del 1% de las multas).
- **Clúster 12**: Actitudes incívicas (menos del 1% de las multas).
- **Clúster 14**: Falta de documentación (menos del 1% de las multas).
- **Clúster 16**: Uso incorrecto de dispositivos de seguridad (menos del 1% de las multas).
  Son clústeres muy pequeños. En todos ellos solo existen dos puntos, (0, 0) y (1, 0), lo que indica que las predicciones son siempre 0, sin embargo, en la realidad hay también vías con 1 multa.

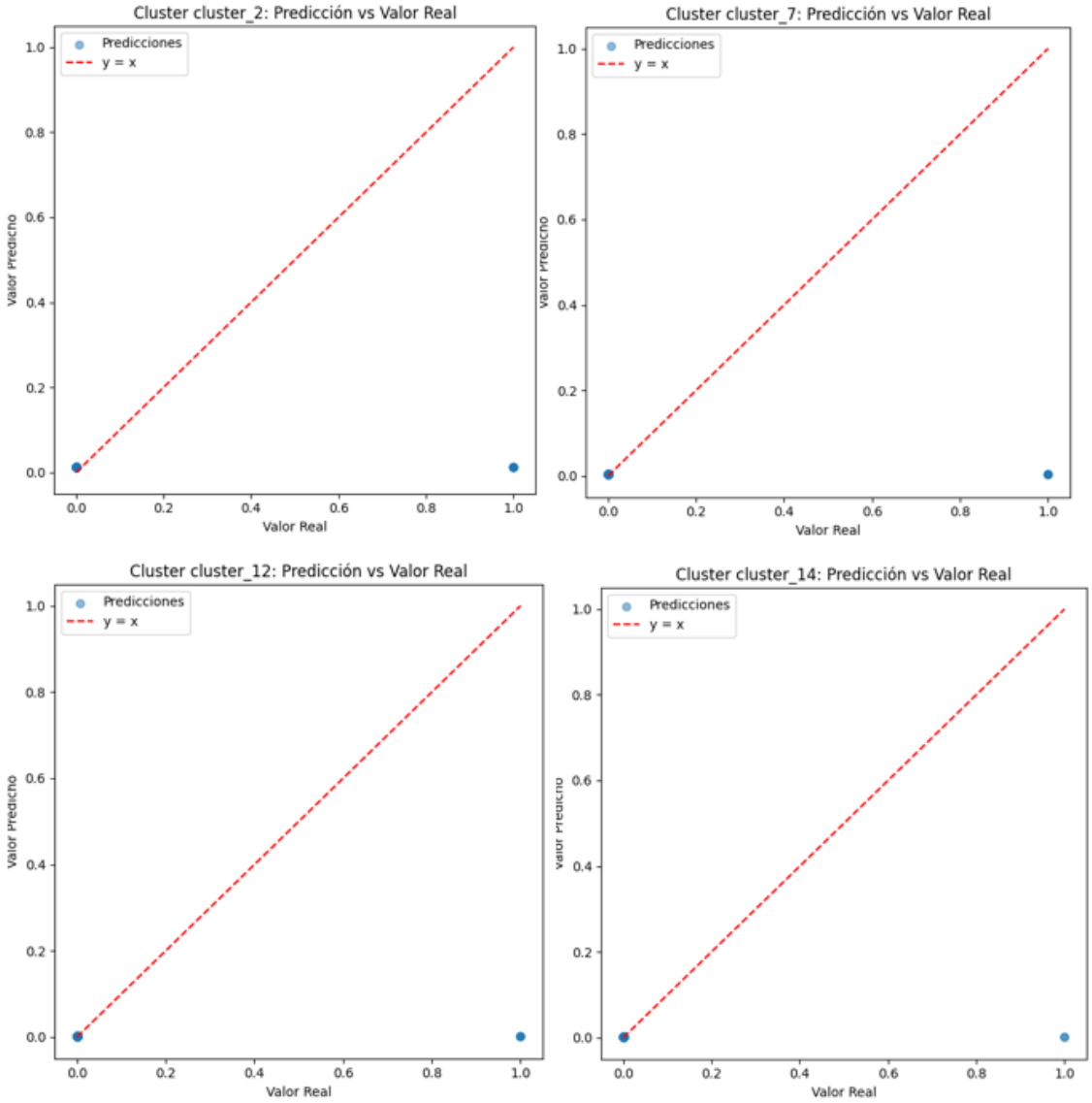


Ilustración 45: Predicción vs Valor Real en Clústeres 2, 7, 12 y 14

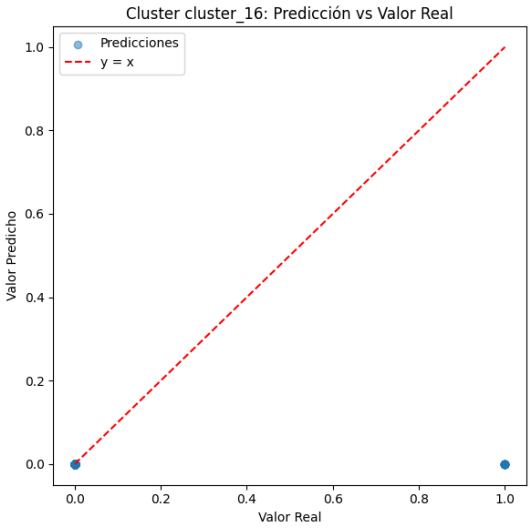


Ilustración 46: Predicción vs Valor real en Clúster 16

- **Clúster 3**: Uso indebido de dispositivos electrónicos (menos del 1% de las multas). El modelo funciona bien para valores bajos, pero tiende a subestimar en valores más altos.

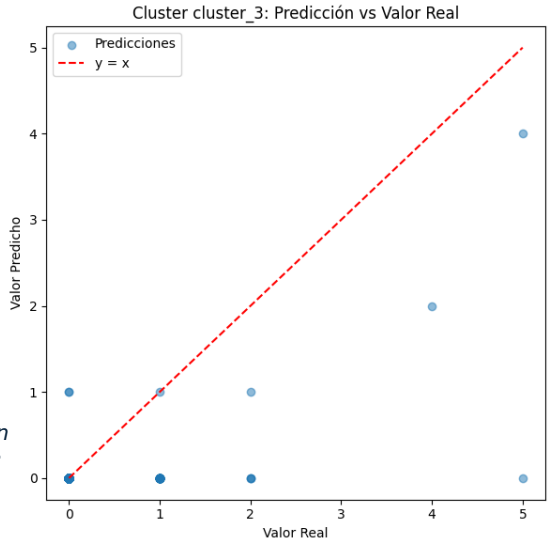


Ilustración 47: Predicción vs Valor real en Clúster 3

- **Clúster 4**: Adelantamientos o cambios de sentido imprudentes (menos del 1% de las multas). Para las vías con 1 multa las predicciones son un poco diversas y existe un valor extremo que subestima, pero en general funciona bien.

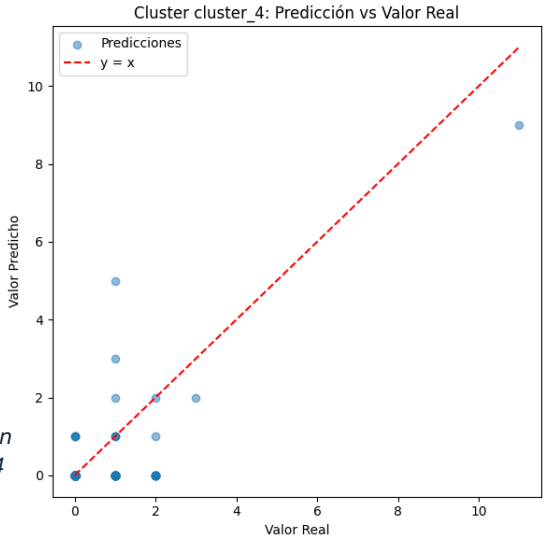


Ilustración 48: Predicción vs Valor real en Clúster 4

- **Clúster 5**: Conducción negligente (aprox. el 8% de las multas). En este clúster hay un rango mucho más amplio de valores reales. El modelo predice bien los valores medios-bajos, pero no siempre es preciso en valores altos.

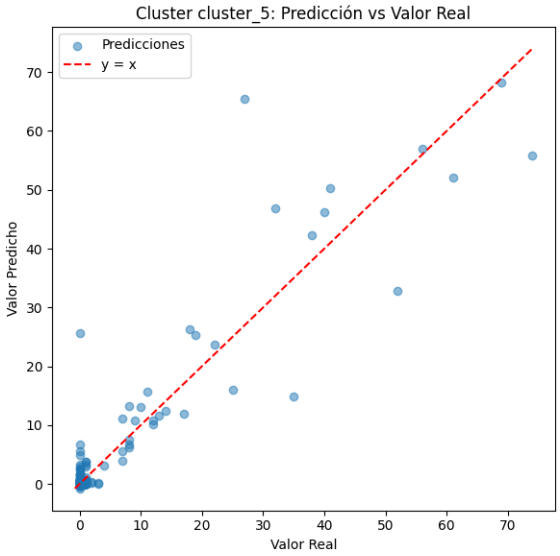


Ilustración 49: Predicción vs Valor Real en Clúster 5

Clúster 6: Exceso de velocidad (aprox. el 35% de las multas). Destaca la enorme amplitud del rango de valores reales. Hay un valor extremo el cual el modelo subestima, pero para el resto de los valores funciona bastante bien.

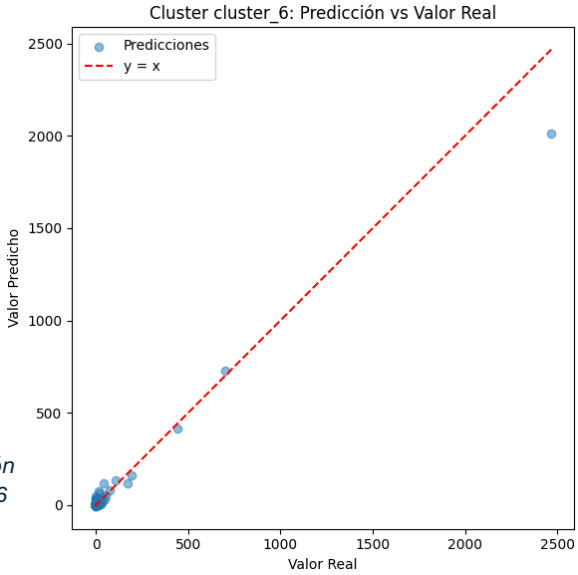


Ilustración 50: Predicción vs Valor real en Clúster 6

- **Clúster 8**: Vehículos de movilidad personal (menos del 1% de las multas).
- **Clúster 9**: Emisión de gases contaminantes (menos del 1% de las multas).
- **Clúster 11**: Abandono del vehículo sin seguridad (menos del 1% de las multas).
  No existen valores reales para estos clústeres y las predicciones son correctas.

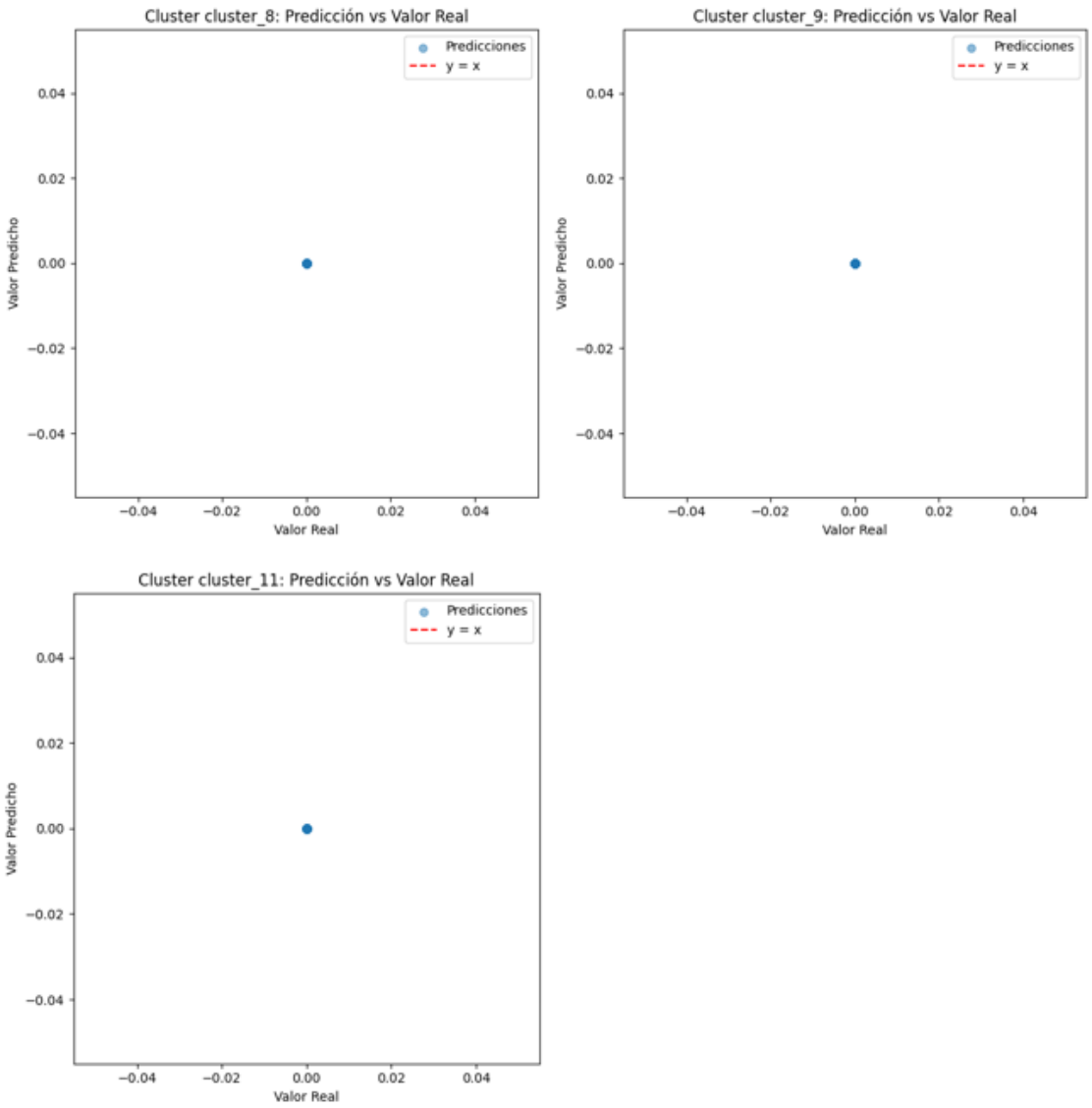


Ilustración 51: Predicción vs Valor real en Clústeres 8,9 y 11

- **Clúster 10**: Conducción bajo efectos del alcohol o drogas (menos del 1% de las multas). En este clúster el modelo tiende a sobreestimar los valores reales a cero y, al contrario, los valores reales 1 y 2 tiende a subestimarlos. De todos modos, la escala es muy pequeña para ambos, por lo que la predicción no es tan mala.

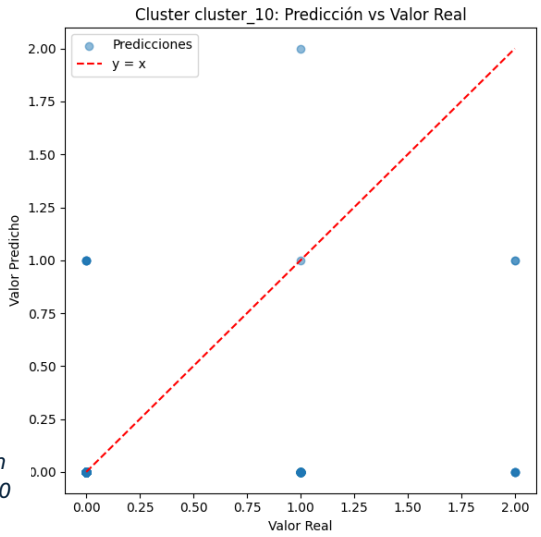


Ilustración 52: Predicción vs Valor real en Clúster 10

- **Clúster 13**: Motocicletas, ciclomotores y bicicletas (menos del 1% de las multas).
- **Clúster 15**: Transporte de mercancías o pasajeros (menos del 1% de las multas).
- **Clúster 17**: Invasión de zona peatonal (aprox. el 1% de las multas). A pesar de haber valores reales de hasta 3 multas por vía, el modelo no predice valores para estos clústeres.

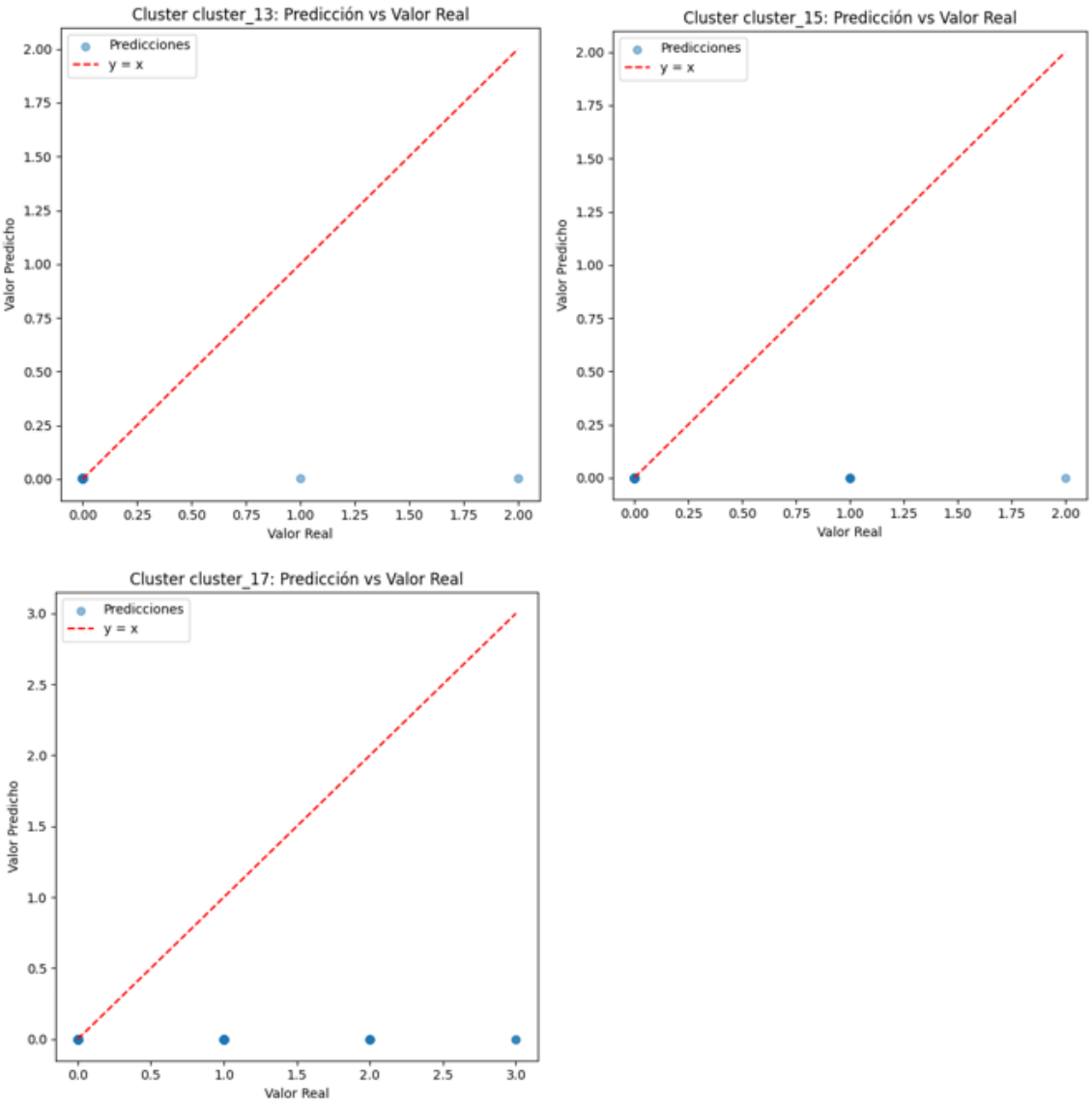


Ilustración 53: Predicción vs Valor real en Clústeres 13, 15 y 17

- **Clúster 18**: Acceso a zona restringida (aprox. el 48% de las multas). Es el clúster más grande, por eso se observa un rango de valores amplio. En el rango de valores reales de 100 a 200, el modelo tiende a sobreestimar, pero para el resto funciona bastante bien.

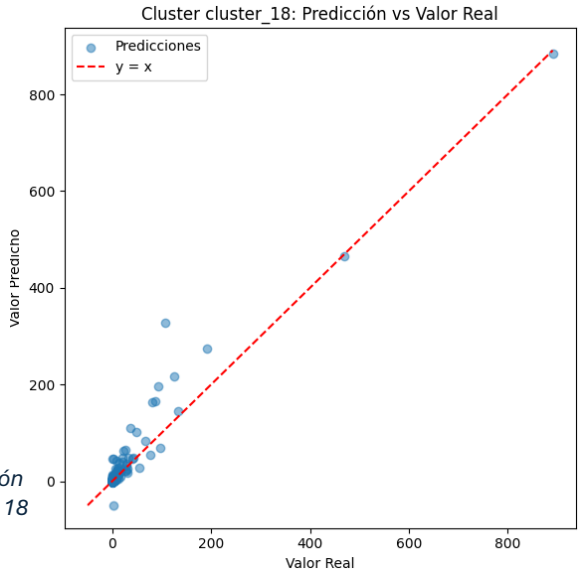


Ilustración 54: Predicción vs Valor real en Clúster 18

Y tras este análisis, se puede concluir que:

- Existe subestimación y sobreestimación en valores reales medios-altos en la escala de cada clúster.
- Hay 8 clústeres en los que las predicciones son cero, pero, aunque la mayoría de los valores reales sean cero, también existen valores de 1, 2 o 3 multas por vía.
- Hay 3 clústeres sin ninguna multa real y con predicciones correctas a cero.
- En clústeres grandes la subestimación/sobreestimación de la predicción supone rangos más amplios, pero en los pequeños tan solo hay diferencias de 1 a 5 multas por vía/mes.

3.7.7. Validación final

Para la validación final con nuevos datos que aún no hubiese visto el modelo se siguieron los siguientes pasos:

1. Se confeccionó un nuevo dataset solo con junio de 2024 simulando el futuro, por lo que, como no se conocen las vías para las que habrá multas en ese futuro, se obtuvieron todas las vías únicas del dataset completo de entrenamiento (todas las vías en las que se produjeron multas de enero de 2022 a mayo de 2024) y se les asignó las features fijas (‘fecha’, ‘estacion’ y ‘clase_via’) y las features agregadas calculadas del mismo modo que en el dataset de entrenamiento.
2. Se volvió a realizar el preprocesamiento y el entrenamiento con el mismo modelo, ajustes, e hiperparámetros, pero incluyendo mayo de 2024 en train en lugar de en test.
3. Se realizaron las predicciones directamente sobre el dataset de junio 2024, empleando los encoder y scaler ya ajustados durante el entrenamiento.

4. Se agregaron los 18 campos de predicciones de clúster al dataset de junio 2024 para su análisis.

Para un análisis más visual de los resultados, se trasladó el CSV resultante a Microsoft Power BI.

En primer lugar, se establecieron rangos de desviación del valor predicho con respecto al real, ya fuesen positivos o negativos, y se contaron cuántas vías había en cada rango para cada clúster:

CLÚSTER 1

c1_dif_real	vías	%TG
Exacto	923	20,23%
Entre 1 y 10	3640	79,77%

CLÚSTER 2

c2_dif_real	vías	%TG
Exacto	4555	99,82%
Entre 1 y 10	8	0,18%

CLÚSTER 3

c3_dif_real	vías	%TG
Exacto	4521	99,08%
Entre 1 y 10	42	0,92%

CLÚSTER 7

c7_dif_real	vías	%TG
Exacto	4560	99,93%
Entre 1 y 10	3	0,07%

CLÚSTER 8

c8_dif_real	vías	%TG
Exacto	4563	100,00%

CLÚSTER 9

c9_dif_real	vías	%TG
Exacto	4563	100,00%

CLÚSTER 13

c13_dif_real	vías	%TG
Exacto	4559	99,91%
Entre 1 y 10	4	0,09%

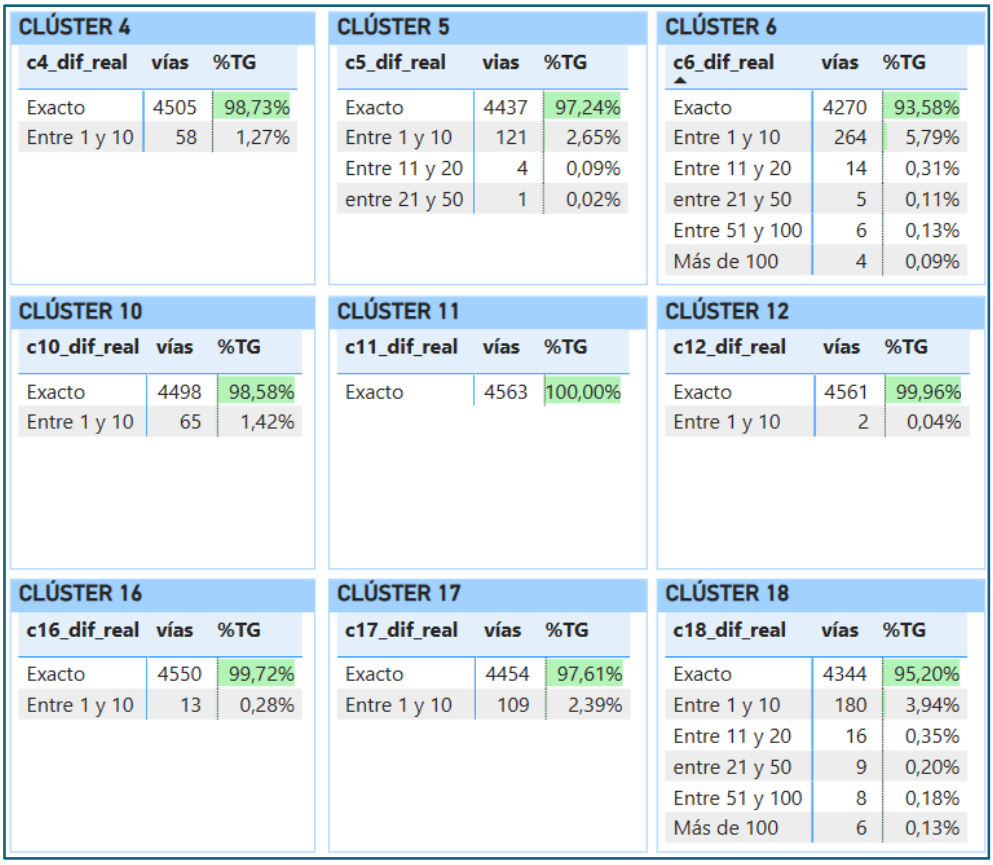
CLÚSTER 14

c14_dif_real	vías	%TG
Exacto	4563	100,00%

CLÚSTER 15

c15_dif_real	vías	%TG
Exacto	4548	99,67%
Entre 1 y 10	15	0,33%

Ilustración 55: Recuento de vías por rangos de exactitud en las predicciones (parte I)



CLÚSTER 4

c4_dif_real	vías	%TG
Exacto	4505	98,73%
Entre 1 y 10	58	1,27%

CLÚSTER 5

c5_dif_real	vias	%TG
Exacto	4437	97,24%
Entre 1 y 10	121	2,65%
Entre 11 y 20	4	0,09%
entre 21 y 50	1	0,02%

CLÚSTER 6

c6_dif_real	vías	%TG
Exacto	4270	93,58%
Entre 1 y 10	264	5,79%
Entre 11 y 20	14	0,31%
entre 21 y 50	5	0,11%
Entre 51 y 100	6	0,13%
Más de 100	4	0,09%

CLÚSTER 10

c10_dif_real	vías	%TG
Exacto	4498	98,58%
Entre 1 y 10	65	1,42%

CLÚSTER 11

c11_dif_real	vías	%TG
Exacto	4563	100,00%

CLÚSTER 12

c12_dif_real	vías	%TG
Exacto	4561	99,96%
Entre 1 y 10	2	0,04%

CLÚSTER 16

c16_dif_real	vías	%TG
Exacto	4550	99,72%
Entre 1 y 10	13	0,28%

CLÚSTER 17

c17_dif_real	vías	%TG
Exacto	4454	97,61%
Entre 1 y 10	109	2,39%

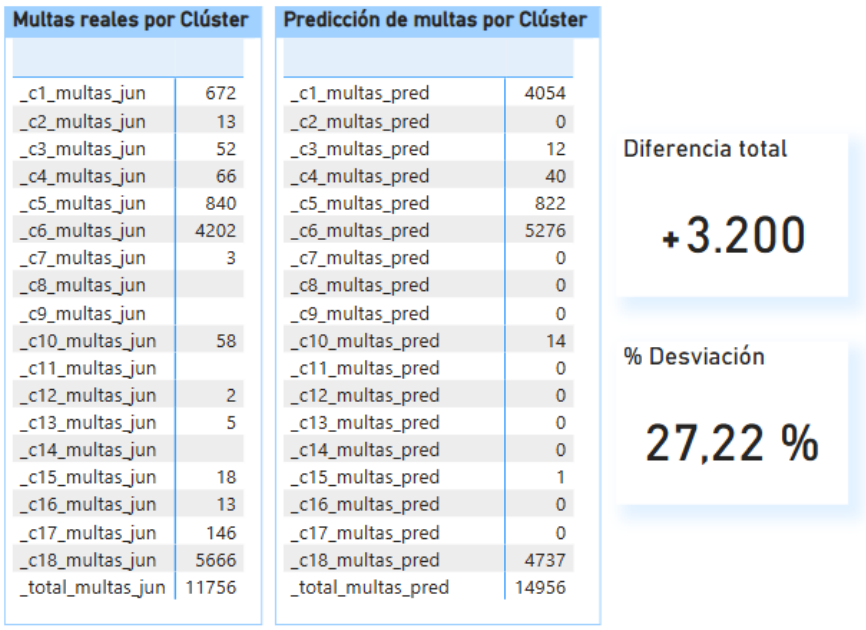
CLÚSTER 18

c18_dif_real	vías	%TG
Exacto	4344	95,20%
Entre 1 y 10	180	3,94%
Entre 11 y 20	16	0,35%
entre 21 y 50	9	0,20%
Entre 51 y 100	8	0,18%
Más de 100	6	0,13%

Ilustración 56: Recuento de vías por rangos de exactitud en las predicciones (parte II)

En la imagen se puede observar que, excepto el clúster 1, todos los demás tienen más del 90% de las vías con una predicción exacta. De hecho, los clústeres 8, 9, 11 y 14 son exactos al 100%. Para el rango “Entre 1 y 10” (1 a 10 multas por encima o por debajo a las reales), exceptuando el cluster 1 que tiene aquí casi el 80% de las vías, los porcentajes oscilan entre 0,04 del clúster 12 al 5,79 del clúster 6. El resto de los rangos están solo presentes en los clústeres 5, 6 y 18 y sus porcentajes son inferiores al 1%. Por lo tanto, la exactitud de las predicciones para cada clúster es bastante buena, exceptuando el clúster 1, que tiende a desviarse en 1 a 10 multas de la realidad.

A continuación, se analizaron el número total de multas predichas con respecto a multas reales por clúster y en total:



Multas reales por Clúster	
_c1_multas_jun	672
_c2_multas_jun	13
_c3_multas_jun	52
_c4_multas_jun	66
_c5_multas_jun	840
_c6_multas_jun	4202
_c7_multas_jun	3
_c8_multas_jun	
_c9_multas_jun	
_c10_multas_jun	58
_c11_multas_jun	
_c12_multas_jun	2
_c13_multas_jun	5
_c14_multas_jun	
_c15_multas_jun	18
_c16_multas_jun	13
_c17_multas_jun	146
_c18_multas_jun	5666
_total_multas_jun	11756

Predicción de multas por Clúster	
_c1_multas_pred	4054
_c2_multas_pred	0
_c3_multas_pred	12
_c4_multas_pred	40
_c5_multas_pred	822
_c6_multas_pred	5276
_c7_multas_pred	0
_c8_multas_pred	0
_c9_multas_pred	0
_c10_multas_pred	14
_c11_multas_pred	0
_c12_multas_pred	0
_c13_multas_pred	0
_c14_multas_pred	0
_c15_multas_pred	1
_c16_multas_pred	0
_c17_multas_pred	0
_c18_multas_pred	4737
_total_multas_pred	14956

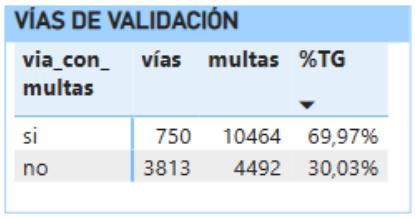
Ilustración 57: Multas totales reales vs predichas

Los clústeres con mayor diferencia son el 1, con 3.382 multas sobreestimadas, el 6 con 1.074 multas sobreestimadas y el 18 con 929 subestimadas. También llama la atención el clúster 17, que no tiene ninguna predicción, pero 146 multas en la realidad.

A nivel global el modelo ha predicho 3.200 multas de más, lo que supone un desvío del 27,22% positivo.

Ahora bien, hay que tener en cuenta un punto clave. Como en el callejero de Madrid hay más de 9.000 vías, para el dataset de validación se usaron todas aquellas que tuvieron multas graves y muy graves durante el periodo que sirvió de entrenamiento, de enero de 2022 a mayo de 2024. En total fueron 4.563 vías distintas.

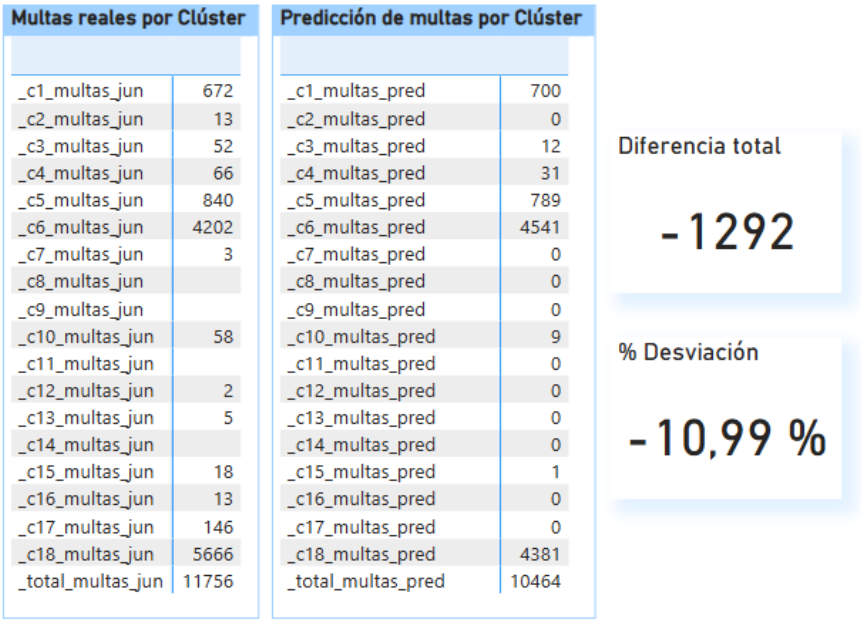
Al cotejar estas 4.563 vías con el dataset de multas de junio, se observa que solo 750 de esas vías tienen multas en la realidad:



VÍAS DE VALIDACIÓN			
via_con_multas	vías	multas	%TG
si	750	10464	69,97%
no	3813	4492	30,03%

Ilustración 58: Vías del dataset de validación con/sin multas reales

Esas 750 vías acumulan un total de 10.464 multas, y al filtrar las tablas de multas reales/predichas por clúster por estas vías, la situación cambia:



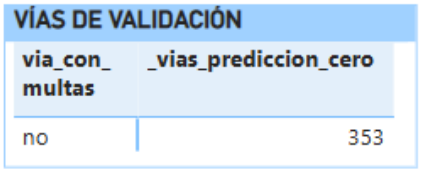
Multas reales por Clúster	
_c1_multas_jun	672
_c2_multas_jun	13
_c3_multas_jun	52
_c4_multas_jun	66
_c5_multas_jun	840
_c6_multas_jun	4202
_c7_multas_jun	3
_c8_multas_jun	
_c9_multas_jun	
_c10_multas_jun	58
_c11_multas_jun	
_c12_multas_jun	2
_c13_multas_jun	5
_c14_multas_jun	
_c15_multas_jun	18
_c16_multas_jun	13
_c17_multas_jun	146
_c18_multas_jun	5666
_total_multas_jun	11756

Predicción de multas por Clúster	
_c1_multas_pred	700
_c2_multas_pred	0
_c3_multas_pred	12
_c4_multas_pred	31
_c5_multas_pred	789
_c6_multas_pred	4541
_c7_multas_pred	0
_c8_multas_pred	0
_c9_multas_pred	0
_c10_multas_pred	9
_c11_multas_pred	0
_c12_multas_pred	0
_c13_multas_pred	0
_c14_multas_pred	0
_c15_multas_pred	1
_c16_multas_pred	0
_c17_multas_pred	0
_c18_multas_pred	4381
_total_multas_pred	10464

Ilustración 59: Multas totales reales vs predichas filtradas por vías con multas reales

Se observa que las diferencias entre clústeres, salvo para el 18, se suavizan, y la desviación total ahora pasa a ser del -10,99%. Esto sugiere que la dificultad radica en que el modelo no puede diferenciar correctamente unas vías de otras en ciertos clústeres, como en el 1, probablemente por falta de patrones sólidos de las features, o bien también podrían ser patrones inesperados de los datos, las llamadas “excepciones que rompen la regla”.

De las 3.813 vías con predicciones que no tienen multas reales, 353 arrojaron predicción cero, por lo que realmente serían 3.460.



VÍAS DE VALIDACIÓN	
via_con_multas	_vias_prediccion_cero
no	353

Ilustración 60: Vías del dataset de validación sin multas reales, pero predicción cero

De esas 3.460 vías, el 95,23% tienen una única multa predicha:

Vías con multas predichas sin multas reales

total_multas_predichas	vías	%TG
1	3295	95,23%
2	81	2,34%
3	16	0,46%
4	23	0,66%
5	12	0,35%
6	7	0,20%
7	6	0,17%
8	2	0,06%
9	2	0,06%
10	3	0,09%
12	1	0,03%
13	1	0,03%
15	1	0,03%
18	2	0,06%
20	1	0,03%
25	1	0,03%
44	1	0,03%
55	1	0,03%
75	1	0,03%
108	1	0,03%
125	1	0,03%
159	1	0,03%
Total	**3460**	**100,00%**

Ilustración 61: Vías del dataset de validación con predicciones, pero sin multas reales

Al filtrar por aquellas que tienen más de 10 multas predichas, quedan solo 13 vías:

Vías con multas predichas sin multas reales

total_multas_predichas	via
12	paseo camoens y valero
13	avenida gran via hortaleza
15	avenida principal
18	avenida sur aeropuerto barajas
18	calle monasterios suso y yuso
20	calle nuestra senora valverde
25	calle vitruvio
44	plaza cortes
55	calle castillo candanchu
75	calle cerro castanar
108	calle san norberto
125	avenida victoria
159	calle gabriela mistral

Ilustración 62: Vías del dataset de validación con más de 10 multas predichas, pero sin multas reales

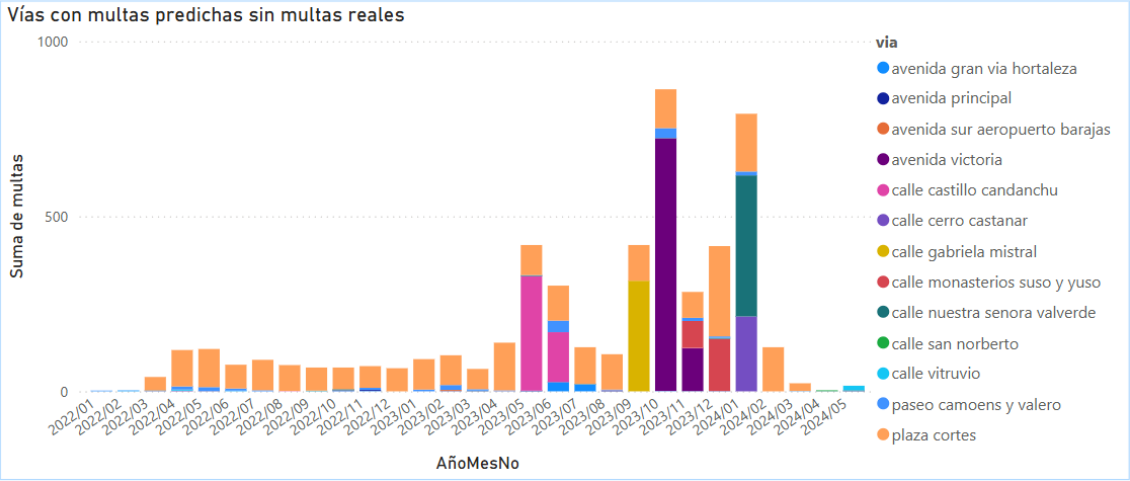


Ilustración 63: Evolución de las multas en las vías mencionadas

Y visualizando la evolución del total multas graves y muy graves en estas vías durante el periodo de tiempo que se usó para entrenamiento, se observa que para varias de ellas hay meses con alto número de multas en el tramo del último año. Teniendo en cuenta que se añadieron como features el promedio de multas en los 6 y 12 meses anteriores, es probable que sean estas las causantes de la predicción, puesto que el resto de features, excepto estación y clase de vía, se basan en datos de los últimos 3 meses, que no tienen apenas representación en el gráfico.

Por otro lado, también aparecen en el dataset de multas de junio 43 vías con 45 multas acumuladas que no están presentes en el dataset de entrenamiento, y este factor no es posible predecirlo. De todos modos, el porcentaje es ínfimo.

VÍAS DE JUNIO NUEVAS

via_nueva	vías	multas	%TG
no	750	11711	99,62%
si	43	45	0,38%

Ilustración 64: Vías nuevas de las multas reales de junio 2024

Para finalizar, se realizó un análisis de las 20 vías con mayor número de multas reales acumuladas en el dataset de junio de 2024 y sus predicciones:

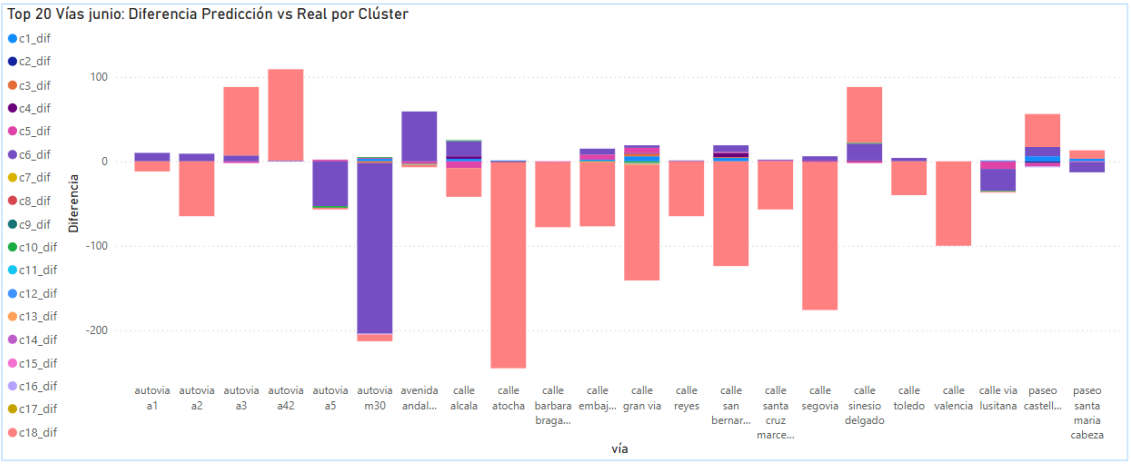


Ilustración 65: Top 20 vías de junio: Predicción vs Real por Clúster

Se observa que son los clústeres 6 y 18 (exceso de velocidad y acceso a zona restringida respectivamente) los que tienen mayor desviación. Hay que recordar que, por un lado, estos fueron los clústeres con mayor RMSE arrojada en la validación de test y, por otro lado, que en el análisis de dispersiones de las predicciones de test solían sobreestimar/subestimar más mientras mayores fuesen los valores reales de multas.

Para ese Top 20 de vías reales de junio, si se atiende el número de multas totales, la mayoría de las predicciones se desvía en más de 10, con un máximo de 245 multas en calle Atocha. Solo autovía A-1 y paseo Santa María Cabeza se aproximan en menos de 10. Además, si se comparan con el Top 20 de vías de las predicciones, las vías marcadas con una flecha roja no aparecen en ambos rankings.

Top 20 Vías reales

vía	total_multas_reales	total_multas_predichas	_desviacion_pred
autovia m30	2583	2369	-214
autovia a1	832	830	-2
autovia a5	625	570	-55
calle alcala	566	552	-14
autovia a2	477	421	-56
calle sinesio delgado	386	474	88
calle gran via	359	229	-130
calle atocha	343	98	-245
calle segovia	279	111	-168
paseo castellana	264	306	42
autovia a42	262	371	109
calle san bernardo	262	154	-108
calle via lusitana	178	140	-38
paseo santa maria cabeza	152	147	-5
calle embajadores	143	79	-64
calle valencia	132	32	-100
calle toledo	128	93	-35
autovia a3	124	210	86
calle barbara braganza	122	44	-78
avenida andalucia	118	171	53
calle reyes	94	30	-64
calle santa cruz marcenado	94	39	-55
Total			**-1053**

Top 20 Vías predichas

vía	total_multas_predichas	Suma de total_multas_reales	_desviacion_pred
autovia m30	2369	2583	-214
autovia a1	830	832	-2
autovia a5	570	625	-55
calle alcala	552	566	-14
calle sinesio delgado	474	386	88
autovia a2	421	477	-56
autovia a42	371	262	109
paseo castellana	306	264	42
calle gran via	229	359	-130
autovia a3	210	124	86
avenida andalucia	171	118	53
calle gabriela mistral	159	0	159
calle san bernardo	154	262	-108
paseo santa maria cabeza	147	152	-5
calle via lusitana	140	178	-38
avenida victoria	125	0	125
calle alfonso xii	114	13	101
calle segovia	111	279	-168
calle san norberto	108	0	108
carretera castilla	100	45	55
Total		**7525**	**136**

Ilustración 66: Top 20 vías reales vs Top 20 vías predichas

4. Conclusiones

En cuanto al análisis de multas graves y muy graves se puede concluir que:

- Existen patrones estacionales en las multas.
- Los importes de multa habituales son 100 y 200€ para multas graves y 500, 600 y 1000 para muy graves.
- Solo suele haber retirada de puntos en multas muy graves, y la cantidad habitual es de 4, 5 o 6 puntos.
- Los denunciantes que imponen más cantidad de multas graves/muy graves son agentes de movilidad y policía municipal. Movilidad radar es el único denunciante que solo impone multas graves/muy graves pero su volumen global es inferior al 1%.
- Los denunciantes Sace, Ser y Medios de captación de imagen solo ponen multas graves/muy graves de un tipo determinado y de importe 200€.
- La mayoría de las multas graves/muy graves tienen lugar por la mañana o por la tarde, en proporción similar, y disminuyen por la noche.
- Tanto la velocidad límite de la vía como la cantidad de velocidad excedida influyen en la sanción de la infracción (importe y puntos).
- Las autovías M-30, A-42 y A-5, junto con calle Alcalá, son las vías con mayor volumen de multas graves/muy graves, siendo sus tipos más frecuentes acceso a zona restringida y exceso de velocidad.
- El tipo de multa tiene un gran peso en la sanción impuesta (importe y puntos).

En cuanto al modelo predictivo construido, se puede concluir que:

- En clústeres pequeños la desviación de la predicción es solo de entre 1 a 5 multas por vía.
- En clústeres grandes el modelo afina mejor en los rangos de valores reales pequeños que en los medios-altos, en los que suele tanto subestimar como sobreestimar, dependiendo del clúster.
- Mientras mayor es el volumen de multas, la desviación de las predicciones se va volviendo mayor.
- Exceptuando el clúster 1, el modelo es capaz de predecir en cada clúster el número exacto de multas para más del 90% de las vías.
- En los clústeres 8, 9, 11 y 14 el modelo predice el número de multas exacto en el 100% de las vías.
- En el clúster 1, la predicción se desvía entre 1 a 10 multas por vía en el 79,77% de las vías, prediciendo el número exacto solo en el 20,23% restante.
- Para el resto de los clústeres, el porcentaje de vías con desviación de entre 1 a 10 multas oscila entre el 0,04% del clúster 12 al 5,79% del clúster 6.
- Solo los clústeres 5, 6 y 18 tienen vías que se desvían del valor real de multas en más de 10, pero el porcentaje de estas vías es inferior al 1% en cada clúster.

- Atendiendo al número total de multas, hay una diferencia de 3.200 multas predichas de más con respecto a las reales de junio, lo que supone un 27,22% de desviación positiva.
- De esas 3.200 multas 45 serían imposibles de predecir, ya que corresponden a nuevas vías aparecidas en junio que nunca han sido vistas en el entrenamiento.
- Se predicen multas en 3.460 vías (4.492 multas en total) que no tienen multas reales en junio, aunque en el 95,23% de ellas la predicción es solo de 1 multa.
- Para las 750 vías que tienen multas reales en junio (excluidas las 45 vías nuevas), el modelo realiza una predicción total de 1.292 multas por debajo, lo que supone una desviación negativa del 10,99%.
- La cantidad de multas predicha en vías con un gran volumen de multas es bastante más inexacta que en vías con menor volumen.

La conclusión, teniendo en cuenta todos los datos, es que el desempeño del modelo cumple bastante bien con los objetivos para los que se creó.

En un escenario hipotético en el que las autoridades competentes decidieran utilizar el modelo para reforzar el control sobre las vías con mayor probabilidad de registrar multas graves o muy graves, podrían centrarse en aquellas vías con más de 10 multas predichas. En ese caso, el modelo solo cometería errores en 13 vías, lo que representa el 0,28% del total de vías utilizadas en el entrenamiento. Incluso si se ampliara el umbral a vías con más de 5 multas predichas, el número de errores ascendería únicamente a 33 vías, lo que equivale al 0,72% de las vías entrenadas.

Además, dispondrían de los tipos de multa potenciales para cada vía, lo cual sería muy interesante a la hora de crear estrategias de mejora en diferentes ámbitos de la circulación y seguridad vial.

En definitiva, el modelo desarrollado no solo permite anticipar las vías de mayor concentración de multas graves o muy graves, sino que también proporciona a las autoridades una herramienta predictiva que puede ayudar a optimizar recursos, reforzar la presencia en determinados puntos o diseñar campañas de concienciación más eficaces según los tipos de infracción prevalentes.

4.1. Posibles mejoras futuras

1. Para un entrenamiento real del modelo habría que tomar la totalidad de los datos o en su defecto, una muestra mayor. Dadas las limitaciones del equipo utilizado, solo se ha tenido en cuenta un 10% aleatorio de cada archivo CSV (cada mes). De esta manera es posible que en los datos de prueba y validación aparezcan valores, sobre todo de vías y tipos de multa, que no estén en el entrenamiento, dificultando las labores de predicción. Además, las distribuciones de multas en los diferentes clústeres (tipos de multa) pueden estar sesgada, por lo que las predicciones obtenidas podrían no ajustarse a la realidad.

2. Para este proyecto se ha usado un único modelo de Machine Learning para todo el dataset. Quizás podrían mejorarse los resultados de las predicciones de ciertos clústeres, como el 1, 6 o 18, si se entrenan diferentes modelos para cada uno de ellos, ya que se pudo observar que los resultados de RMSE para un mismo clúster diferían de un modelo a otro.

3. Otra opción para mejorar el modelo podría derivar del desbalanceo de clústeres, ya que hay unos muy grandes, como el 5, 6 y 18, y el resto son muy pequeños. Se podría solucionar agrupando clústeres pequeños, aunque se perdería visibilidad de los motivos de multa. También se podría probar a desglosar los más grandes en otros más pequeños.

4. En cuanto a las features utilizadas para entrenar el modelo, sería una buena estrategia calcular un mayor número de features agregadas, como la suma de multas en los últimos 6 y 12 meses, además de los promedios, o incluso buscar nuevas features en datasets externos que puedan aportar mayor diferenciación entre unas vías y otras, de manera que el modelo sea capaz de captar patrones más complejos y afinar así las predicciones en las vías.

5. Como se comentó durante el proyecto, de cara a añadir nuevos datos al modelo, sería conveniente construir un modelo supervisado de clasificación para asignar a cada nuevo valor de 'hecho' el clúster adecuado y no tener que repetir de nuevo todo el proceso de clusterización.

6. Con el paso del tiempo también sería necesario actualizar el callejero, por si van apareciendo vías nuevas.

7. Por último, también se podría estudiar la manera de añadir las coordenadas a todas las vías, bien sean las de un punto céntrico de cada una de ellas o algún cálculo como el promedio de los diferentes puntos principales que abarcan la vía completa, y realizar un análisis geoespacial para tener una perspectiva más visual de las zonas con mayor concentración de multas.

5. Referencia Bibliográfica

Portal de datos abiertos del Ayuntamiento de Madrid (Multas de circulación: detalle):

- https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=fb9a498a6bdb9410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default

Portal de datos abiertos del Ayuntamiento de Madrid (Callejero Oficial del Ayuntamiento de Madrid):

- https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?page=2&vgnextoid=b3c41f3cf6a6c410VgnVCM2000000c205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default

Código Python:

- **Apuntes**: MDS Área 2 Módulo 3, Lenguajes de Programación en Data Science, unidad 1.
- **Libro**: "Curso Intensivo de Python, Introducción Práctica a la Programación Basada en Proyectos" - Eric Matthes, Anaya.
- **Páginas web**:
  - Módulo re: https://docs.python.org/es/3.13/library/re.html
  - Módulo glob: https://docs.python.org/es/3.13/library/glob.html#module-glob
  - Clases: https://docs.python.org/es/3.13/tutorial/classes.html

Machine Learning:

- **Libro**: "Python Machine Learning, Aprendizaje automático y aprendizaje profundo con Python, Scikit-learn y TensorFlow" - Sebastián Raschka y Vahid Mirjalili, Marcombo
- **Página web:** https://www.datacamp.com/es/tutorial/machine-learning-python

Scikit-learn:

- **Página web:** https://scikit-learn.org/stable/index.html

SBERT:

- **Página web:** https://sbert.net/

SpaCy:

- **Página web:** https://spacy.io/

UMAP:

- **Apuntes**: MDS Área 2 Módulo 5, Diseño de un Modelo Escalable, página 95
- **Página web:** https://umap-learn.readthedocs.io/en/latest/basic_usage.html

DBSCAN:

- **Apuntes**: MDS Área 2 Módulo 5, Diseño de un Modelo Escalable, página 82
- **Página web:** https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

Optuna:

- **Página web:** https://optuna.org/

RapidFuzz:

- **Página web:** https://www.datacamp.com/es/tutorial/fuzzy-string-python
- **Página web:** https://rapidfuzz.github.io/RapidFuzz/

Validación cruzada temporal (TimeSeriesSplit):

- **Página web:** https://codecut.ai/cross-validation-with-time-series/

Random Forest:

- **Apuntes**:
  - MDS Área 2 Módulo 5, Diseño de un Modelo Escalable, página 41
  - MDS Área 3 Módulo 6, Herramientas y Librerías, página 35
- **Página web:** https://cienciadedatos.net/documentos/py08_random_forest_python

MultiOutputRegressor:

- **Páginas web:**
  - https://scikit-learn.org/stable/modules/multiclass.html
  - https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html

LightGBM:

- **Apuntes**: MDS Área 3 Módulo 6, Herramientas y Librerías, página 51
- **Páginas web**:
  - https://lightgbm.readthedocs.io/en/latest/
  - https://docs.aws.amazon.com/es_es/sagemaker/latest/dg/lightgbm-hyperparameters.html

XGBoost:

- **Páginas web**:
  - https://xgboost.readthedocs.io/en/stable/index.html
  - https://www.datacamp.com/es/tutorial/xgboost-in-python
  - https://www.ibm.com/es-es/think/topics/xgboost

CatBoost:

- **Páginas web**:
  - https://catboost.ai/docs/en/
  - https://docs.aws.amazon.com/es_es/sagemaker/latest/dg/catboost-hyperparameters.html