



# SISTEMA DE ANÁLISIS Y PREDICCIÓN DE CONTAMINACIÓN LUMÍNICA

TRABAJO FIN DE GRADO  
Manual Técnico

Grado en Ingeniería Informática  
(doble especialidad: Computación e Ingeniería Computadores)

**Autor:** Antonio Gómez Giménez

**Directores:** Ezequiel Herruzo Gómez  
Fco. Ramón Lara Raya

Córdoba



UNIVERSIDAD DE CÓRDOBA



# **Índice:**

<b>1. Introducción.</b>	<b>1</b>
<b>2. Definición del problema:</b>	<b>4</b>
2.1. Definición del problema Real	4
2.2. Definición del problema Técnico	4
<b>3. Objetivos:</b>	<b>7</b>
<b>4. Antecedentes:</b>	<b>10</b>
4.1. Modelos computacionales	10
4.2. Contaminación lumínica.	11
4.3. Sistemas de adquisición de datos.	12
4.3.1. Espectrorradiómetro HD 30.1.	12
4.3.2. Fotoradiómetro HD2402.	15
<b>5. Restricciones:</b>	<b>20</b>
5.1. Factores Dato	20
5.2. Factores estratégicos	20
<b>6. Recursos:</b>	<b>23</b>
6.1. Recursos humanos	23
6.2. Recursos software	23
6.3. Recursos hardware	24
6.4. Otros tipos de recursos	25
<b>7. Especificación de requisitos:</b>	<b>27</b>
7.1. Requisitos de usuario	27
7.2. Requisitos funcionales	28
7.3. Requisitos no funcionales	29
7.4. Requisitos de información	30
<b>8. Metodología:</b>	<b>32</b>
8.1. Elección de características	32
8.2. Adquisición de los datos	39
8.2.1. Tipo de Luminaria	45
8.2.2. Altura de la Luminaria	58
8.2.3. Flujo Lumínico de la Luminaria	59
8.2.4. Temperatura Correlada con el Color	60
8.2.5. Iluminancia en el Suelo	62
8.2.6. Espectro de la luz	63
8.2.7. Color del Suelo	64
8.2.8. Reflectancia del Suelo	65
8.2.9. Iluminancia encima de la Luminaria	67
8.2.10. Flujo Lumínico Superior a Luminaria y FHSi	68

8.3. Preprocesamiento de los datos y conclusiones sobre la elección de características	71
8.4. Creación del sistema de análisis y predicción de contaminación lumínica	82
8.4.1. Tipos de modelos	82
8.4.1.1. Problemas de clasificación.	82
8.4.1.2. Problemas de regresión.	83
8.4.2. Modelos elegidos para usar en el sistema	84
8.4.2.1. SVR:	84
8.4.2.2. Árbol de decisión:	85
8.4.2.3. RandomForest:	85
8.4.2.4. Regresión Lineal y SGDRegressor:	86
8.4.3. Tipos de errores escogidos	87
8.4.3.1. MSE	87
8.4.3.2. MAE	87
8.4.4. Planteamiento y creación del sistema	88
8.4.4.1. Servicios del sistema	90
8.4.4.1.1 mysql	90
8.4.4.1.2 modelo_prediccion_contaminacion_luminica	90
8.4.4.1.2.1 Insertar en BD	92
8.4.4.1.2.2 Entrenar	93
8.4.4.1.2.3 Predecir a partir de modelo	102
8.4.4.2. Programas auxiliares	104
8.4.4.2.1. Preprocesamiento	104
8.4.4.2.2. Ejemplo Cliente	105
8.4.4.2.3. Generador Gráficas	106
8.4.4.3. Simulación de uso y resultados obtenidos	107
<b>9. Pruebas:</b>	<b>116</b>
<b>10. Conclusiones y futuras mejoras:</b>	<b>122</b>
<b>11. Bibliografía</b>	<b>127</b>
<b>Agradecimientos</b>	<b>132</b>
<b>Anexos</b>	<b>134</b>
<b>Anexo I. Manual de usuario.</b>	<b>136</b>
1. Instalación:	136
1.1. Instalación Docker:	136
1.2. Descarga de github	137
1.3. Descargas auxiliares para programas de apoyo	138
2. Explicación de las carpetas	139
3. Uso de docker	142
4. Peticiones get	143
4.1. Petición: insertar patrones	143
4.2. Petición: entrenar	143

4.3. Función: predecir	144
5. Uso del sistema	145
<b>Anexo II. Manual de código.</b>	<b>150</b>
1. Docker Compose	150
2. Dockerfile	153
3. main.py	155
4. preprocesamiento.py	166
5. generador_grafica.py	168
6. ejemplocliente.py	171



## 1. Introducción.

El uso de las tecnologías de la información y las comunicaciones y su utilización en nuevos entornos y sistemas está cambiando gran parte de los planteamientos que se tenían hasta el momento de su incorporación a dichos entornos y sistemas. Este es el caso del sistema que se plantea en este proyecto donde, por un lado, se trabajará en la **adquisición de datos relacionados con la contaminación lumínica** que se está produciendo mediante los nuevos equipos y sistemas de iluminación y, por otro lado, se trabajará con esos datos aplicando **técnicas de inteligencia artificial** para establecer mecanismos que permitan mejorar la iluminación de los entornos evitando los efectos perjudiciales que la contaminación lumínica puede provocar en dichos entornos.

La iluminación artificial inadecuada tiene consecuencias negativas en su entorno. Su principal efecto es el aumento del brillo del cielo nocturno, lo cual dificulta seriamente las investigaciones astronómicas y puede causar daños a ecosistemas, provocando alteraciones en los ciclos vitales y en los comportamientos de especies animales y vegetales con hábitos de vida nocturnos. Además, el consumo energético se ve innecesariamente incrementado, originando un aumento de los costes económicos y de la producción de contaminantes atmosféricos. Otros impactos negativos recaen en la calidad ambiental de las zonas habitadas, ya que aumenta la intrusión lumínica en el ámbito privado de las personas, provocando molestias tales como fatiga visual, ansiedad y alteraciones del sueño. También dificulta a la población la observación del cielo nocturno.

Teniendo en cuenta que la luz se transmite por la atmósfera a distancias que pueden superar los 100 km desde el lugar en el que se genera, si no se toman medidas, este tipo de contaminación podría acabar con la calidad de nuestro cielo nocturno, de tal forma que se impide su explotación como recurso científico, cultural y económico.

Se entiende por “**Contaminación lumínica**” la emisión de flujo luminoso, por fuentes artificiales de luz constituyentes del alumbrado nocturno, con intensidades, direcciones o rangos espectrales inadecuados para la realización de las actividades previstas en la zona alumbrada.

Se ha demostrado que la luz azul que emite una fuente de luz en el rango visible, durante la noche, es la más perjudicial para la biodiversidad y para las observaciones astronómicas.

Este tipo de emisiones de luz están muy presentes en las luminarias basadas en tecnologías LED, las cuales han experimentado un notable aumento en el alumbrado público de muchos municipios, que han encontrado en esta tecnología una reducción del consumo energético y, por ende, de un ahorro económico para sus arcas.

La normativa establece diferentes zonificaciones lumínicas, en función del uso de los espacios (espacios naturales, espacios urbanizables residenciales, industriales, comerciales o de gran densidad de tránsito de personas y vehículos). Según cada uno de estos espacios, se admiten unos índices de iluminación basados, no sólo en niveles de Iluminancia y Luminancia, sino también en otros que toman en cuenta el índice de reproducción cromática, así como el índice espectral G, que mide la cantidad de radiación azul.

Por otro lado, los modelos predictivos que pueden realizar los sistemas computacionales actuales, basados en el campo del aprendizaje automático, constan de una serie de técnicas donde se entrena al modelo con una serie de datos, de tal forma que una vez entrenado, al recibir unos parámetros de entrada da un valor de salida que pretende predecir un resultado futuro. Estos sistemas de modelos predictivos permiten tener una ayuda a la hora de tomar decisiones en el entorno o sistema en el que se aplique.

En la actualidad hay diversidad de modelos, por ejemplo, **redes neuronales** (Bishop & Bishop, 2007, #), **rbf** (Berzal, 2018, #), **k-vecinos** (Tan, Pang-Ning, 1 enero 2005, #), **svm** (John Shawe-Taylor & Nello Cristianini, 23 marzo 2000, #), etc. Para afrontar este problema se realizará un análisis de los datos que se disponen y los objetivos de consecución que permitirá diseñar un modelo que nos aporte los mejores resultados. Para ello, en principio se probará el entrenamiento con distintos tipos de modelos como pueden ser redes neuronales, **rbf** u otro tipo de modelo para obtener aquel modelo que ofrezca mejores resultados. Se prevé que se necesitará ajustar los modelos con ciertos parámetros para buscar la mejor combinación en el problema planteado.



## 2. Definición del problema:

### 2.1. Definición del problema Real

Tras la introducción realizada anteriormente, se pretende definir más concretamente el problema a solucionar. Como se comentó, el problema principal era realizar una **investigación sobre la contaminación lumínica para averiguar qué datos son importantes para la adquisición de los mismos y la creación de un modelo que permita predecir la contaminación lumínica.**

En la actualidad, la **contaminación está muy presente**, y en nuestro caso, la contaminación lumínica también. La contaminación lumínica es un problema a tener en cuenta ya que elimina o empeora parcialmente la contemplación del cielo nocturno, siendo un recurso científico, cultural y económico en peligro. También, puede ser que afecte a otros tipos de problemas con cierta relevancia, como vimos anteriormente con la atracción de los insectos a los pueblos, y por lo tanto, a la fauna y flora. Este tipo de problemas pueden ser bastante perjudiciales para los pueblos, y a la larga incluso para las ciudades.

Por lo tanto, el problema a solucionar es **ser capaces de detectar esa contaminación lumínica** cuanto antes para evitar problemas futuros y en el caso de que ya exista esa contaminación lumínica, darse cuenta de que existe y eliminarla.

### 2.2. Definición del problema Técnico

Este al ser un tema tan complejo, se optó por solucionar el siguiente problema, siendo un subproblema del anterior. El problema a resolver es ver **qué parámetros afectan en mayor medida a la contaminación del cielo nocturno**, este es un problema que, si se consigue solucionar, proporcionará información relevante para otros proyectos futuros como solucionar el problema explicado anteriormente.

Por lo tanto el problema a resolver es, con ciertos parámetros de entrada que se definirán en puntos posteriores, crear **un modelo que nos ayude a la toma de decisión a la hora de evaluar si hay contaminación lumínica o habrá**

**contaminación lumínica.** Todo el tema de la investigación y el sistema a realizar donde se encontrará el modelo, se describe en puntos posteriores.

Respecto al **funcionamiento**, se espera que sea un sistema sencillo donde se introducen datos de entrada simulando un patrón y el sistema predice un resultado a partir de ese patrón con el modelo entrenado, de esta forma se obtendrá un valor que indicará si hay contaminación lumínica o no.

Respecto al **usuario** que va dirigido el sistema, en principio cualquier persona pero principalmente será más útil el sistema a aquellas personas que trabajan en el campo porque entenderán mejor los parámetros de entrada y como deben de modificarlos para evitar que exista contaminación lumínica.

Respecto al **tiempo de respuesta**, se espera que a la hora de entrenar el modelo, el sistema sea relativamente lento (en un rango entre 10 y 50 minutos), pero a la hora de predecir un patrón con el modelo ya entrenado se espera que el resultado se obtenga casi inmediatamente, permitiendo al usuario final probar distintas combinaciones sin una demora excesiva por parte del sistema.

Respecto al **mantenimiento**, como este proyecto es un TFG no se pretende darle mantenimiento pero no se descarta en un futuro añadirle ciertas mejoras como incluir más variedad de parámetros o incluso probar otros algoritmos o incluir más patrones de entrada para ver como varían los resultados del sistema.

Respecto a la **competencia**, como se va a ver en el apartado de antecedentes, existen algunos casos de intentos de investigación para solucionar o dar su visión respecto al problema de mosquitos que se dirigen a los pueblos. Este TFG pretende reducir el problema a uno más sencillo y más entendible para ver qué parámetros influyen más en la contaminación lumínica, permitiendo establecer unas bases sólidas para proyectos futuros.



### 3. Objetivos:

Los objetivos que se pretenden cumplir para llegar a solucionar el problema identificado anteriormente son los siguientes:

- **OB-01-> Comprensión y establecimiento de parámetros** para el análisis de la contaminación lumínica.
- **OB-02-> Capacidad de obtención de datos y de desarrollo de una metodología clara para adquisición de los mismos**, datos relacionados con la contaminación lumínica (distancias, estructura, luminarias, ...)
- **OB-03-> Adecuación de parámetros de contaminación lumínica** para el laboratorio, adaptación al trabajo de campo.
- **OB-04-> Diseño y adaptación de sistema de adquisición**, almacenamiento y formato de los datos captados.
- **OB-05-> Análisis de los datos obtenidos** y establecimiento de **parámetros determinantes** para los indicadores de contaminación lumínica.
- **OB-06-> Desarrollo de un modelo predictivo de contaminación lumínica** basado en tecnologías de aprendizaje automático (siendo este el objetivo más importante). Para este objetivo se pueden encontrar algunos sub-objetivos como:
  - **Modularidad** a la hora de implementar el sistema.
  - **Comparativa de diferentes modelos** para ver qué modelo ofrece mejores resultados al problema identificado con anterioridad.
  - **Velocidad del sistema**, tanto a la hora de entrenar cómo predecir (en la medida de lo posible con unos tiempos aceptables)
- **OB-07-> Validación del modelo** de contaminación lumínica.

Los puntos presentados con anterioridad se le van a asignar una prioridad alta a la hora de realizar este **TFG**, se podrían incluir otros objetivos y metas adicionales para mejorar el resultado de este **TFG**, pero se prefiere incluir una cantidad de objetivos reducida para completar la mayoría de los mismos e incluso todos.

Si se consigue realizar algún avance o mejora en este **TFG** que no se haya comentado en esta lista, se comentará en el apartado de conclusiones.

Cabe destacar que también se pretenden lograr otro tipos de objetivos que no están tan ligados a los objetivos a conseguir sobre como solucionar el problema identificado, sino más bien, **objetivos a un nivel más personal**.

Algunos de ellos son:

- Ser capaz de **adentrarse y resolver un problema de una rama diferente a la de los estudios realizados** como puede ser todo el tema de la contaminación lumínica y cómo funcionan las luminarias.
- Ser capaz de **aprender nuevas tecnologías** que permitan mejorar la modularidad del modelo, como puede ser aprender **Docker** para crear contenedores para cada servicio, permitiendo instalar todo el sistema para cualquier tipo de máquina.
- Ser capaz de **reorganizar el desarrollo del TFG**, frente a diferentes tipos de problemas que vayan surgiendo, para poder lograr completar todos los objetivos, o en su defecto, la mayoría de los mismos.
- Tras terminar el TFG, conseguir **adquirir diferentes conocimientos y características tras el proceso**, aparte de aprender sobre luminaria, docker, python, etc. Sino otras características como mayor constancia, mayor capacidad para relacionarme con diferentes personas que puedan mejorar el resultado del proyecto, mayor esfuerzo.

A la hora de nombrar objetivos se pueden obtener muchos, ya sea a nivel de objetivos del proyecto, como objetivos a nivel personal. Los nombrados anteriormente son los considerados más importantes a mi parecer.



## 4. Antecedentes:

Respecto a los antecedentes que se van a mostrar, se podría considerar que hay tres vertientes, siendo la primera **modelos anteriormente creados**, la segunda, los **estudios sobre la contaminación lumínica**, y, siendo la tercera, **sistemas de adquisición de datos**. Este proyecto se basa en una combinación de ambas, por ello, se buscarán casos similares creados con anterioridad.

### 4.1. Modelos computacionales

Respecto a los modelos computacionales de aprendizaje automático podemos encontrar muchos tipos que se han creado con anterioridad. En este proyecto se usarán uno o varios de estos modelos ya que, dependiendo de los datos, estos afectarán al rendimiento del modelo.

Antes de realizar el modelo, se realizará un análisis y tratamiento de los datos, para ello podemos encontrar algunos programas como **Weka** (Witten et al., 2016, #), o **Matlab** (*MATLAB - El Lenguaje Del Cálculo Técnico - MATLAB & Simulink*, n.d.) que se pueden encontrar ya desarrollados.

Finalmente, respecto a los modelos ya desarrollados, podemos encontrar algoritmos como:

El modelo de **K-vecinos** (Tan, Pang-Ning, 1 enero 2005, #), es un modelo que se basa principalmente en utilizar los “*k*” puntos “más cercanos” (vecinos más cercanos teniendo en cuenta la distancia) para realizar la clasificación.

Las **redes neuronales** de distintos tipos (dependiendo del objetivo) (Bishop & Bishop, 2007, #), las redes neuronales buscan copiar a los animales que son capaces de reaccionar de forma adaptativa a los cambios en su entorno externo e interno, y utilizan su sistema nervioso para realizar estas conductas. Por ello se busca representar el concepto de neurona y los enlaces entre las mismas. Dependiendo de la estructura que se utilice para llevar a cabo este objetivo, podemos encontrar distintos tipos de redes neuronales como rbf donde su función de activación es de este tipo o redes de perceptrón multicapa para problemas de clasificación con función de activación de tipo sigmoide, todo dependerá del objetivo que busquemos y que tipo red neuronal se adapta más al problema.

También podemos encontrar **máquinas de vectores soporte** (John Shawe-Taylor & Nello Cristianini, 23 marzo 2000, #), cuyo principal objetivo es, dado un conjunto de ejemplos de entrenamiento, construir un hiperplano “w” como superficie de decisión. De tal forma que la separación de las dos clases sea máxima (principio de generalización). Para este problema no encajaría este tipo de modelo ya que no disponemos de dos clases, a no ser que se realice algún tipo de modificación para adaptarlo.

## 4.2. Contaminación lumínica.

Respecto a la contaminación lumínica, se puede decir que es un tema bastante tratado. De hecho, en España hay incluso una organización llamada “*Red española de investigación en contaminación lumínica, REECL*”, que busca interconectar a los distintos investigadores del campo de la contaminación lumínica. Esta organización se encuentra financiada por el gobierno y busca solucionar ciertos problemas ligados a la contaminación lumínica. Para poder llevar a cabo estas investigaciones, disponen de más de 40 fotómetros operativos en la base de datos de **Night Sky Brightness de STARS4ALL** (*Contaminación Lumínica – Stars4All*, 2015).

Entre los parámetros que se analizan para estudiar la contaminación lumínica existente en ciertos entornos destacan los siguientes: **Illuminancia**, Temperatura de color correlacionada **CCT**, Coordenadas Tricromáticas, **Índice de reproducción cromática**, Irradiación UVA, Irradiación **UVB** e Irradiación **UVC**.

El parámetro **TCC** (temperatura de color), indica la percepción del color de la luz por el ojo humano, pero no mide la cantidad de luz azul emitida, por ello, se analizará el parámetro de **índice G** (*Inicio Índice Espectral G*, n.d.), que mide la cantidad de radiación azul que emite una fuente de luz en el rango visible. De esta forma se nos permite analizar la luz azul para ver si se cumplen los límites de ésta para determinar si existe contaminación lumínica.

También se analizarán otros parámetros como la **iluminancia** o la **luminancia**. La **luminancia** se refiere a la cantidad de luz que percibe el ojo humano tras reflejarse sobre la superficie de cualquier objeto y se mide en candela por metro cuadrado. Respecto a la **iluminancia**, se refiere a la incidencia de la luz sobre una superficie (no la cantidad de luz emitida por la fuente) y se mide en lux. (*BOE-A-2008-18634 Real Decreto 1890/2008, De 14 De Noviembre, Por El Que Se Aprueba El Reglamento De Eficiencia Energética En Instalaciones De Alumbrado Exterior Y Sus Instrucciones Técnicas Complementarias EA-01 a EA-07.*, 2021)

También podemos ver en otras páginas incluso cómo medir la contaminación lumínica y posibles soluciones (GUTIERREZ, n.d.)

Se han podido encontrar diversos artículos donde se trata este tema, por ejemplo, en el siguiente artículo (Olsen, R. N, 2014, #), se comenta un modelo donde la economía se ve afectada por la contaminación lumínica, dejando de lado los riesgos de la vida salvaje y de los humanos.

Se han encontrado, además, artículos donde se comentaban el crecimiento de la contaminación lumínica o incluso en el siguiente artículo (Donners, M., 2018, #), se comenta mediante un modelo, como los insectos se ven atraídos por distintos tipos de fuentes de luz de colores, pudiéndose llegar a relacionar con el tema de la contaminación lumínica.

En definitiva, se trata de un tema con **bastante proyección** y sobre el que hay mucha gente interesada ya que como se puede observar existen, incluso, numerosas líneas de investigación que de una u otra forma lo están tratando, aunque no se haya encontrado ninguna realización o desarrollo sobre sistemas o modelos que se asemejen a lo que se pretende realizar en este trabajo de fin de grado.

#### 4.3. Sistemas de adquisición de datos.

Como es conocido, existen gran variedad y diversidad de sistemas sensores para la captación y adquisición de todo tipo de información externa a los dispositivos electrónicos e informáticos.

Entre ellos nos encontramos con numerosos tipos de sistemas sensores que podrían ser utilizados para determinar los **indicadores sobre radiaciones** que interesan en este TFG. En concreto nos vamos a centrar en los siguientes:

##### 4.3.1. Espectrorradiómetro HD 30.1.

El **HD30.1** es un instrumento fabricado por Delta Ohm para el análisis espectral de la luz visible y ultravioleta. El instrumento ha sido diseñado combinando la máxima flexibilidad de uso, accesibilidad y facilidad de uso.

La siguiente figura muestra este dispositivo, denominado comercialmente como indicador de registro de datos **HD30.1**.



Imagen 4.3.1.1: dispositivo HD30.1

Se compone de dos elementos, sensores de campo espectral, conectados por medio de un cable: el indicador de registro de datos **HD30.1** y los sensores de medición HD30.S1 (rango espectral entre 380nm-780nm) y **HD30.S2** (rango espectral entre 220nm-400nm).

La siguiente figura muestra cómo son estos sensores de medición HD30.S1 y HD30.S2



Imagen 4.3.1.2: sensores de medición HD30.S1 y HD30.S2

El indicador de registro de datos **HD30.1**, con sistema operativo linux, procesa y gestiona los datos. Cuenta con una gran pantalla táctil a color, que permite una visualización sencilla de las medidas y facilidades de almacenamiento, como muestran las siguientes figuras.



Imagen 4.3.1.3: indicador de registro de datos

Los espectros y los parámetros derivados se pueden guardar tanto en la memoria interna (150 MB), como en una memoria externa (micro SD-card o una memoria USB). El formato de exportación es compatible con los programas más comunes para el análisis y procesamiento de datos. Además de guardar los datos, el software le permite guardar imágenes de los gráficos.

Las principales magnitudes de interés foto-radiométricas se calculan directamente de HD30.1 a través del software suministrado. El rangopectral analizado varía dependiendo del sensor utilizado para medir:

- Región del Espectral Visible (380nm-780nm) con el sensor **HD30.S1**.
- Región del Espectro Ultravioleta (220 nm-400 nm) con el sensor **HD30.S2**.

Los sensores de medición son intercambiables y calibrados (el archivo de calibración se almacena dentro de cada sonda). El sensor HD30.S1 analiza el espectro visible (380nm-780nm) y calcula las siguientes magnitudes foto-colorimétricas: Illuminancia [lux], Temperatura de color correlacionada CCT [K], Coordenadas Tricromáticas [x,y] (CIE 1931) o [u',v'](CIE1978), CRI (índice de reproducción cromática, R1...R14, Ra) , PAR [ $\mu$ mol/ft<sup>2</sup>/sm<sup>2</sup>].

El sensor HD30.S2 analiza la bandaespectral ultravioleta (220nm-400 nm) y calcula las siguientes magnitudes radiométricas: Irradiación UVA (W/m<sup>2</sup>), Irradiación UVB (W/m<sup>2</sup>) e Irradiación UVC (W/m<sup>2</sup>).

Ambos sensores tienen una vista de la entrada equipada con una nueva generación de difusor que optimiza la respuesta de acuerdo con la ley del coseno y para no introducir ninguna deformación espectral. Los datos relativos a la calibración de cada sonda se almacenan en la memoria permanente y se leen por el indicador de instrumento.

El sistema funciona con baterías internas (recargables, 3.7V, 6.6Ah) o conectados a su fuente de alimentación (SWD06), que tiene la doble función de alimentar la unidad y cargar la batería. La duración de la batería con el instrumento encendido es de aproximadamente 10 horas, lo que puede aumentar las condiciones particulares de uso.

#### 4.3.2. Fotorradiómetro HD2402.

El **HD2402** es un fotorradiómetro datalogger portátil para realizar mediciones de radiaciones ópticas de acuerdo con la Directiva Europea 2006/25/CE y con el Decreto Ley núm. 81 fechado 9 de abril de 2008.

El instrumento consiste en una serie de sensores para cubrir distintas porciones del espectro y en un pequeño LÁSER utilizado para indicar la fuente analizada. Los distintos sensores trabajan en los siguientes rangos de espectro:

- Sensor fotométrico para medir la iluminancia (luxómetro) dentro del rango de espectro 380÷780 nm.
- Sensor radiométrico para la banda ultravioleta (220÷400 nm) con factor de peso espectral  $S(\lambda)$ .
- Sensor radiométrico para la banda de los rayos UVA (315÷400 nm).
- Sensor radiométrico para la banda 400÷700 nm (azul) con factor de peso espectral  $B(\lambda)$ .
- Sensor radiométrico para la banda IR (700÷1300 nm) con factor de peso espectral  $R(\lambda)$ .
- Sensor de termopila para la medición de la iluminancia en el infrarrojo, rango espectral 400÷2800 nm.

El **HD2402** es un instrumento que puede ser alimentado conectándolo a un ordenador, recibiendo la alimentación directamente del puerto USB o a través de un alimentador externo con salida USB.

Se puede configurar el **HD2402** (calendario, fecha, hora, hora de inicio y duración del logging), descargar y analizar los datos almacenados, y adquirir datos en tiempo real. Una vez configurado, el datalogger puede ser desconectado del ordenador y conectado a su alimentador para adquirir y almacenar los datos.

El instrumento está equipado con un botón para permitir cualquier inicialización y parada manual del almacenamiento cuando no esté conectado al ordenador. Un indicador de **LED** en la parte posterior indica el estado de adquisición del instrumento. La siguiente figura muestra una imagen de este dispositivo de adquisición de datos.



Imagen 4.3.2.1: dispositivo de adquisición de datos HD2402

Como se puede observar existen numerosos sensores disponibles en el dispositivo. A continuación, se muestra una imagen identificando dichos sensores.

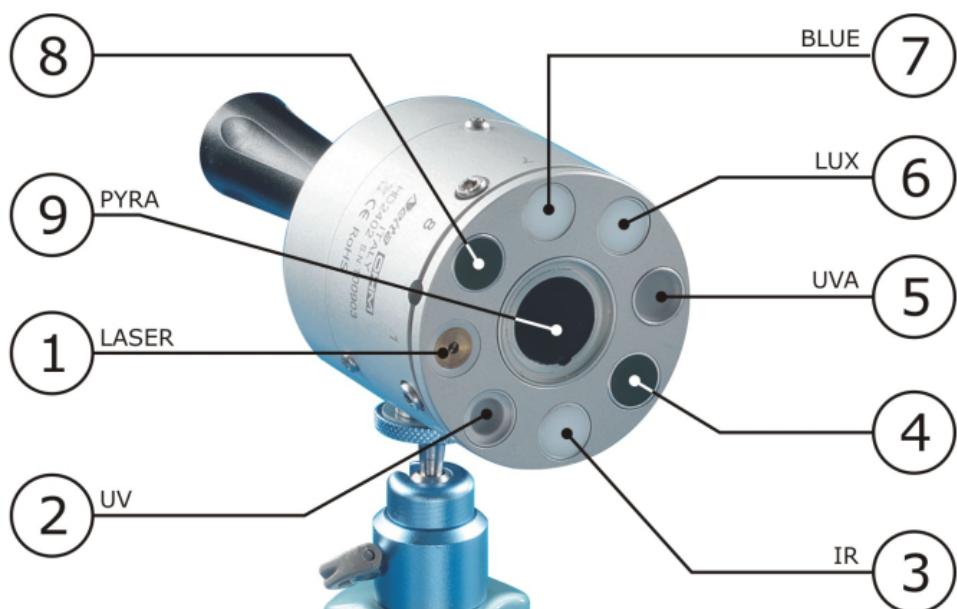


Imagen 4.3.2.2: identificación sensores

Donde:

1. LÁSER LED
2. Sensor radiométrico para la medición de la banda ultravioleta (220 ÷ 400 nm)
3. Sensor radiométrico para la medición de la banda NIR (700÷1300 nm)
4. No usado
5. Sensor radiométrico para la medición de la banda UVA (315÷400 nm)
6. Sensor fotométrico para la medición de la luz visible (Luxómetro)
7. Sensor radiométrico para la medición de la banda AZUL (400÷600 nm)
8. No usado
9. Sensor de termopila para la medición de la banda NIR-FIR (400÷2800 nm)

A continuación, se muestra un ejemplo de uso de este dispositivo mediante la aplicación que aporta el fabricante. Para hacer esto y poder analizar los datos y realizar el informe de evaluación se debe haber elegido el origen de los datos, a partir de las opciones anteriores, se debe haber seleccionado un archivo de medidas y asociado a una fuente la ventana de un proyecto. Tras esto aparecerá un interfaz similar al que se muestra en la siguiente figura.

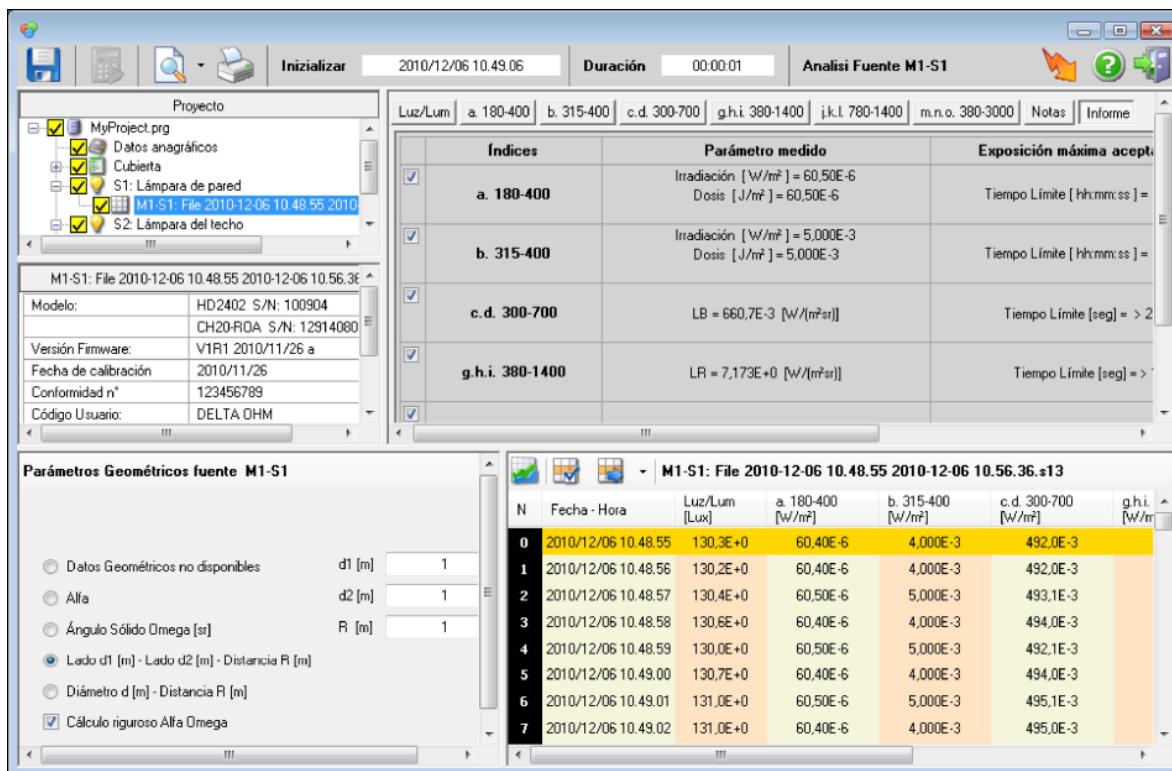


Imagen 4.3.2.3: ejemplo de uso de HD2402

Respecto a la calibración de los sensores, indicar que estos instrumentos son calibrados en la empresa y no requieren otras operaciones por el usuario. Cada sensor que forma el **HD2402** está calibrado singularmente con modalidades distintas como se indica a continuación:

**Luxómetro (Canal 6)**, la calibración se realiza comprando con el luxómetro muestra de segunda línea del laboratorio metrológico Delta OHM; el medio de comparación es una lámpara de incandescencia con temperatura de color de 2856K (Iluminante A).

**Radiómetro UV (Canal 2)**, la calibración se realiza comprando con el radiómetro muestra de segunda línea del laboratorio metrológico Delta OHM; la calibración ocurre con luz monocromática de 270nm obtenida a la salida de un doble monocromador y usando como fuente una lámpara de Xe-Hg.

**Radiómetro UVA (Canal 5)**, la calibración se realiza comprando con el radiómetro muestra de segunda línea del laboratorio metrológico Delta OHM; la calibración ocurre con luz monocromática de 365nm obtenida filtrando una lámpara de Xe-Hg con filtro interferencial de 365nm.

**Radiómetro AZUL (Canal 7)**, la calibración se realiza comprando con el radiómetro muestra de segunda línea del laboratorio metrológico Delta OHM; la calibración ocurre con luz monocromática de 440nm obtenida filtrando una lámpara halógena con filtro interferencial de 440nm.

**Radiómetro IR (Canal 3)**, la calibración se realiza comprando con el radiómetro muestra de segunda línea del laboratorio metrológico Delta OHM; la calibración ocurre con luz monocromática de 680nm obtenida filtrando una lámpara halógena con filtro interferencial de 680nm.

**Piranómetro (Canal 9)**, la calibración se realiza comprando con el piranómetro muestra usando la luz producida por una lámpara halógena. La luz es perpendicular a la superficie de la termopila. La calibración ocurre de acuerdo con la regulación ISO 9847.



## 5. Restricciones:

### 5.1. Factores Dato

Los factores datos son todas aquellas restricciones que están adheridas al problema de forma implícita y que no son modificables o elegibles, es decir, son impuestas por la naturaleza del problema. Para nuestro problema en concreto nos encontramos las siguientes restricciones:

- **El tiempo.** Al ser un proyecto de TFG, el proyecto tiene una fecha límite que no puede sobrepasarse, siendo la mayor prioridad del proyecto. Por lo tanto, si es necesario rechazar posibles mejoras del proyecto como adquisición de material a largo plazo, se prescindirá para cumplir esta restricción.
- **El dinero.** Como se ve en el apartado de antecedentes, los sensores son muy útiles con pantallas integradas para ver la información captada, etc. Pero su precio es bastante elevado dependiendo del sensor. Por ello, se pretende buscar el material disponible en la uco para satisfacer este apartado, ya que para realizar las pruebas de laboratorio también será necesario realizar gastos (en menor medida) para realizar la adquisición de datos.

### 5.2. Factores estratégicos

Respecto a los factores estratégicos, son aquellas restricciones que pueden ser modificables. Son alternativas a tener en cuenta a la hora de resolver el problema, de tal forma, que se escogerán aquellas que sean más beneficiosas para el proyecto. Las restricciones escogidas para este proyecto son:

- **Plataforma objetivo.** De entre todas las opciones posibles que se pueden encontrar, se desea que el sistema se pueda desplegar en cualquier dispositivo de distribución linux. Para ello, el sistema debe ser modular y fácilmente despegable, para poder llegar a cumplir esta restricción, se pretende usar docker. Para lograrlo, se elige linux en vez de otros sistemas operativos por su sencillez y por la facilidad de uso a la hora de usar ciertos lenguajes como python junto con docker y ciertas librerías como scikit.

Respecto a usar docker, se pretende usar docker en vez de otras opciones como kubernetes. Por su facilidad de uso y su claridad a la hora de trabajar con los contenedores.

- **Plataforma de desarrollo.** Como se pretende usar linux en la plataforma objetivo, a la hora de realizar el desarrollo del sistema, se pretende usar linux, para así mantener cierta constancia.
- **Lenguajes y librerías.** Respecto al lenguaje, se pretende usar python respecto a otros, por su sencillez en su uso y por la cantidad de librerías existentes que permiten aumentar la velocidad en el desarrollo del proyecto. Respecto a librerías, se pretende usar sobre todo, scikit learn. Esta librería incluye tanto modelos, preprocesamiento, y todo aquello necesario para realizar el estudio y predicción de datos ya sean problemas de clasificación o regresión. Al ser una librería tan potente, es muy necesaria para realizar el proyecto de una forma rápida y eficiente.



## 6. Recursos:

### 6.1. Recursos humanos

- **Autor:** Antonio Gómez Giménez. Alumno de quinto curso del grado de ingeniería informática, doble rama de computadores y computación.
- **Director:** Ezequiel Herruzo Gómez. Profesor del Departamento de Ingeniería Electrónica y de Computadores.
- **Director:** Fco. Ramón Lara Raya. Profesor del Departamento de Ingeniería Eléctrica.

### 6.2. Recursos software

- **Sistema Operativo:** Todo el desarrollo de sistema se realizará en linux, en concreto Ubuntu 20.04.4 LTS. Para la documentación, el sistema operativo no será relevante, ya que se realizará desde drive, pudiendo usar linux o windows para desarrollarla.
- **Desarrollo de software:** Para el desarrollo del sistema se utilizará el lenguaje de programación python (Python, n.d.), por su facilidad de uso y aparte por la gran cantidad de librerías que nos podemos encontrar. En concreto, se usará sobre todo la librería scikit-learn (scikit, n.d.), ya que proporciona ventajas a la hora de realizar modelos y trabajar con los mismos. Todo este desarrollo se realizará de forma modular usando docker (Docker, n.d.), ya que permite la creación de contenedores (y uso de imágenes ya existentes) para asignar servicios en los mismos, de tal forma que podemos crear imágenes de esos contenedores y desplegarlos en cualquier arquitectura de forma independiente al sistema operativo, por ejemplo.
- **Documentación:** Se realizará en drive usando google docs, principalmente por su facilidad de uso y por que ofrece la posibilidad de trabajar online desde cualquier dispositivo y lugar.

- **Editor de texto:** Para poder realizar el código, se usará como editor de textos sublime text, por ser muy liviano, fácil de instalar, con resaltos de todo tipo de lenguaje con colores para visualmente detectar fallos a simple vista, etc.  
Como apunte adicional, es un editor de textos que he utilizado desde que empecé a programar y no hay ningún inconveniente a la hora de utilizarlo en este proyecto.
- **Repositorio y control de versiones:** El control de versiones se realizará en git, usando GitHub. Ya que este, no es de pago.
- **Base de datos:** Como se necesita un servicio de almacenamiento de datos para guardar la información, se usará *mysql*, principalmente por su facilidad de uso y su sencillez.

### 6.3. Recursos hardware

- **Dispositivo de desarrollo y despliegue:** Se realizará en un portátil Toshiba Satellite L50D-C-13P de 8G de ram, procesador AMD A10-8700p radeon r6 y una gráfica AMD Radeon r6 graphics. Se ha escogido este portátil ya que es el portátil de uso propio del autor del tfg, de tal manera que se podían abaratar costes así.
- **Sensores:** Respecto a los sensores usados, se han usado dos (ambos pertenecientes a la uco, para evitar costes). Estos son el *uprtek premium* y *luxómetro Ix9621 philips*.

El uprtek premium (Castaneyra, 2021) es un espectrómetro de mano con un coste de alrededor de 6000€ que nos ofrece una serie de parámetros ante distintos tipos de iluminación.

Respecto al luxómetro Ix9621 philips se usará para medir la iluminancia que llega a un punto, permitiendo extraer ciertos parámetros para el modelo.

## 6.4. Otros tipos de recursos

- **Estructura de extracción de datos:** es una estructura creada a mano, cuyo objetivo es la adquisición de datos a nivel de laboratorio, se explicará con más detalle en dicho apartado. Es necesario añadir recursos para su montaje, como telas, guantes, bridás, pinzas, tornillos, etc. También se usarán distintos tipos de luminarias que se explicarán más adelante.
- **Libros:** A la hora de aprender sobre este tema (desconocido para el autor del tfg), se han suministrado ciertos libros de mucha ayuda para entender mejor todo lo relacionado con el tema de parámetros y conceptos, siendo el libro “Guía técnica de adaptación de las instalaciones de alumbrado exterior al decreto 375/2010 de 3 de agosto” (*Guía Técnica De Adaptación De Las Instalaciones De Alumbrado Exterior Al Decreto 375/2010 De 3 De Agosto*, 2013) y el libro “Principios básicos del alumbrado” (*Fundamentos Sobre La Generación De La Luz Y El Alumbrado*, 2011).



## 7. Especificación de requisitos:

En este apartado se pretende realizar un análisis donde se describen los requisitos que debe cumplir el proyecto, cumpliendo con los **factores Dato** y **estratégicos** comentados en puntos anteriores. Estos requisitos se dividen en 4 tipos: **requisitos de usuario, requisitos funcionales, requisitos no funcionales y requisitos de información.**

### 7.1. Requisitos de usuario

Los **requisitos de usuario** son aquellos requisitos que vienen dados por el usuario y condicionan el proyecto, de tal forma que, estos requisitos deben coincidir con la funcionalidad del proyecto que se está desarrollando. Para mayor claridad, a cada requisito se le nombrará con la nomenclatura **RU** añadiendo el número del requisito.

**RU - 1:** El sistema debe ser capaz de insertar datos en una base de datos creada a partir de un fichero csv.

**RU - 2:** El sistema debe ser capaz de entrenar todos los modelos una vez que tenga datos en la base de datos.

**RU - 3:** El sistema debe ser capaz de detectar cual de todos los modelos es el mejor y que este se almacene para su uso.

**RU - 4:** El sistema debe ser capaz de predecir la iluminación superior a partir de un patrón utilizando el mejor modelo.

**RU - 5:** Los servicios, tanto de la base de datos como el propio servicio que ofrece el sistema, deben encontrarse aislados en contenedores de docker.

**RU - 6:** El arranque del sistema deberá hacerse con docker-compose.

**RU - 7:** El sistema debe contener una base de datos donde almacenar los datos con los que se pretende trabajar.

## 7.2. Requisitos funcionales

Los **requisitos funcionales** son aquellos que hacen referencia a la funcionalidad concreta que realiza cada parte del sistema a desarrollar. Para este caso, la nomenclatura será **RFUN** seguido del número del requisito.

**RFUN - 1:** La base de datos debe ser creada con mysql.

**RFUN - 2:** La base de datos debe tener creada una tabla donde se encuentren todos los atributos con los que se pretende trabajar (a la hora de usar estos atributos no se tiene que ver obligado a usarlos todos).

**RFUN - 3:** Los servicios deben estar encapsulados en docker para poder usarlo dentro de un docker-compose.

**RFUN - 4:** El sistema a la hora de entrenar debe ser capaz de almacenar los mejores modelos para cada combinación y el mejor modelo en general. Cabe destacar que deberá almacenar el preprocesamiento realizado para los datos ya que es necesario para quitar ese preprocesamiento a posteriori.

**RFUN - 5:** El sistema a la hora de entrenar deberá almacenar en distintos ficheros de texto los resultados obtenidos para cada modelo para cada tipo de combinación.

**RFUN - 6:** El servicio ofrecido por el docker donde se encuentran los modelos, debe responder peticiones “get”, siendo inserción, entrenamiento y predicción.

**RFUN - 7:** La devolución del resultado de la predicción debe encontrarse en formato JSON para mayor facilidad de uso.

**RFUN - 8:** El sistema una vez se inicia a través del docker compose, debe ser resistente a fallos en sus peticiones, evitando que caiga todo el sistema.

**RFUN - 9:** El sistema dará información de las peticiones que recibe a través de la consola con la que fue iniciado, además cada vez que termine de realizar una petición devolverá un mensaje de éxito o error (se excluye el caso de la predicción que devuelve un JSON).

### 7.3. Requisitos no funcionales

Los **requisitos no funcionales** hacen referencia a aquellos requisitos que no tienen que verse implicados con el funcionamiento general del sistema, ni de cada una de sus partes o de la información que maneja el sistema. Para este caso, la nomenclatura será **RNFUN** seguido del número del requisito.

**RNFUN - 1:** El sistema global está formado por dos servicios (la base de datos y el servicio que contiene los modelos), estos deben ser claramente diferenciables, siendo de esta manera un sistema modular y fácilmente mantenable.

**RNFUN - 2:** El sistema al ser modular (al realizarse con docker), debe permitir mejoras en el futuro y capacidad de añadir nuevos servicios. Esto permite que el sistema sea un sistema escalable.

**RNFUN - 3:** Al introducir ficheros que no son csv el sistema debe detectarlos y enviar un mensaje que avise del problema.

**RNFUN - 4:** Cuando una petición se ha realizado de forma errónea, debido a los parámetros introducidos, el sistema debe devolver un mensaje que avise de dicho problema.

**RNFUN - 5:** Cada vez que se vuelva a lanzar el entrenamiento, deben de eliminarse todos los tipos de modelos guardados con anterioridad, el caso de mejor modelo, sus respectivos preprocesamientos y los ficheros de texto plano con la información de cada prueba y ser sustituidos por los nuevos resultados.

## 7.4. Requisitos de información

Por último, se comentan los **requisitos de información**, estos hacen referencia a la información que manejará el sistema que se pretende desarrollar. Estos requisitos especificarán qué tipo de información necesitará el sistema para funcionar correctamente. Para este caso, la nomenclatura será **RINF** seguido del número del requisito.

**RINF - 1:** En la base de datos, se almacenará exclusivamente la información relacionada con los patrones extraídos en la adquisición de los datos. Puede darse la casualidad de que existan combinaciones de patrones que den los mismos resultados. Es decir, pueden existir patrones repetidos.

**RINF - 2:** Los resultados que predice el modelo, no serán almacenados en la base de datos, para evitar introducir errores a futuros modelos.

**RINF - 3:** El fichero original con los datos de la adquisición datos no debe ser modificado, por eso, esos datos se pasan a una base de datos para poder trabajar con los mismos.

**RINF - 4:** Cada vez que se vaya a trabajar con datos de la base de datos para entrenar, se extraerán los datos necesarios a un fichero csv y a partir de ese fichero se trabajará.



## 8. Metodología:

### 8.1. Elección de características

Para poder realizar una correcta elección de características, se recurrió a los dos libros comentados anteriormente (*Guía Técnica De Adaptación De Las Instalaciones De Alumbrado Exterior Al Decreto 375/2010 De 3 De Agosto, 2013*) y (*Fundamentos Sobre La Generación De La Luz Y El Alumbrado, 2011*), ya que los conocimientos sobre este campo para el autor del tfg eran inexistentes.

Tras la lectura de estos libros se llegó a conclusiones muy interesantes, pero antes es necesario explicar conceptos básicos de este campo y ciertas características para comprenderlo mejor.

Primeramente se van a explicar las mediciones de la luz, estas son:

- **Flujo luminoso:** Cantidad de luz emitida por una fuente. La unidad del flujo lumínico es el lumen (lm).
- **Intensidad luminosa:** Cantidad de luz por segundo en una dirección determinada. La unidad de la intensidad luminosa es la candela (cl).
- **Iluminancia:** Cantidad de luz que llega a una superficie. La unidad de la iluminancia son los lux o lúmenes por metro cuadrado (lm/m<sup>2</sup>).
- **Luminancia:** Cantidad de luz radiada por unidad de área aparente en una dirección determinada. La unidad de la luminancia es la candela por metro cuadrado (cd/m<sup>2</sup>).

Las principales técnicas para medir estos valores son:

- **Flujo luminoso:** Principalmente el flujo luminoso se mide con un instrumento complejo, llamado “esfera de Ulbricht”. Este instrumento es una esfera hueca pintada de blanco mate que permite que todos los puntos de la esfera coincida el flujo lumínico con la iluminancia para que a través de un hueco se pueda medir la iluminancia y por tanto el flujo lumínico.

- **Intensidad luminosa:** La intensidad luminosa se puede medir con un aparato llamado *Goniofotómetro*, este aparato se basa en los principios de fotografía para, a partir de distintos ángulos, medir la intensidad luminosa.
- **Iluminancia:** La iluminancia se puede medir con un sensor llamado *luxómetro*. De los cuatro tipos de instrumentos vistos, este sería el más sencillo de conseguir, ya que es un sensor y no un instrumento de grandes características como los anteriores.
- **Luminancia:** La luminancia se puede medir con un *luminancímetro*. Es un aparato muy complejo que permite medir con un ángulo de sólo 1º a partir de una distancia de 1m hasta el infinito.

Tras ver las mediciones de la luz y como medirlas, podemos entender y comprender otros tipos de datos y conceptos de este campo que pueden ser interesantes comentar.

**Tipos de luminarias.** Actualmente hay muchos tipos de luminarias, pero a medida que avanza el tiempo, se están reduciendo a lámparas led, ya que los beneficios que dan son muy superiores al resto. Su consumo es menor, el coste de las mismas es menor, la iluminación proporcionada es muy buena, la contaminación lumínica al cielo nocturno es menor, etc.

Aun así, se nombran los distintos tipos que hay sin entrar en mucho detalle (en los libros viene mucha más información de cada tipo), para que quede documentado:

- Vapor de sodio (alta y baja presión)
- Halogenuros metálicos (cerámicos y cuarzo)
- Vapor de mercurio
- Fluorescencia
- Inducción
- LED

Los tipos de luminarias no se reducen únicamente a los tipos, sino que también hay otros parámetros a tener en cuenta como la orientación de la lámpara (la altura y posición de la misma), esto influirá en la cantidad de flujo lumínico que va directamente al cielo nocturno y cuanto va hacia el suelo. Esto es importante para ver cuánto se está aprovechando bien una luminaria.

También hay que tener en cuenta la **óptica** de la misma, el reflector que tiene (si es que tiene), ya que dirige la luz y la estructura de la luminaria, por ejemplo si está tapada por encima.

Este trabajo de fin de grado se centra en la contaminación lumínica, aún así, es interesante saber que hay otro objeto de estudio respecto a la luminaria con **el problema de la luz intrusiva** que se cuela a vivienda molestando el sueño de las personas. En parte está ligado a la contaminación lumínica ya que estos problemas suceden por la ineficiencia de la colocación del alumbrado o de la estructura de la luminaria. Ya que una mala orientación de la luminaria puede hacer que la luz vaya al cielo nocturno o se enfoque a una ventana de una vivienda. Además, aquellas lámparas como por ejemplo las luminarias exterior de bola, están siendo retiradas por los mismos motivos. Ya que la luz emitida no es controlada y se dirige en todas las direcciones.

Otros aspectos interesantes a tener en cuenta es el **espectro** (pudiendo ser luz fría o luz cálida). La luz es una forma de energía que se manifiesta como una radiación electromagnética. La longitud de la onda es la única forma de diferenciarlas, de tal forma que se relaciona con el color. Si la longitud de onda es más pequeña el color es más frío y si la longitud de onda es más grande el color es más cálido. El ojo humano solo es capaz de ver entre 380 y 780 nanómetros.

El espectro es muy interesante cuando se comenta la **reflectancia** y el **color**, ya que afecta directamente en el **índice del rendimiento del color**. Los seres humanos, vemos los colores de los objetos gracias a que la luz incide en los mismos, de tal forma que si la luz del sol se refleja en un objeto azul, podemos observar su color.

¿Pero qué pasaría si la luz que incide en un objeto de color azul no disponga de suficiente longitud de onda de color azul?

Dicho objeto se vería de color gris. Por lo tanto, puede darse el caso de que las luces que podrían ser ambas blancas, no tengan el mismo **IRC**. Ya que una luz puede ser el sol (es la mejor luz ya que tiene todas las longitudes de onda) o una luz blanca creada con mezclas de distintas longitudes de onda.

Aparte de todo lo explicado anteriormente, la **reflectancia** es muy importante ya que dependiendo de ésta, la cantidad de flujo lumínico que se refleja en el suelo implica que llegue al cielo nocturno.

La reflectancia dependerá del material, del color (en concreto la **temperatura correlada del color** que se mide en K) y del tipo de luminaria (si es fría o caliente, por lo explicado anteriormente).

En el punto de adquisición de los datos se explicará cómo se ha obtenido los índices de reflectancia para los patrones.

Respecto al espectro, también existe un tema bastante polémico respecto al **índice espectral G o luz azul**, (*Inicio Índice Espectral G*, n.d.) este campo aún no está demasiado desarrollado. Existen estudios que demuestran que el color importa para la vida de los seres humanos y en la contaminación lumínica, demostrando que hay que evitar fuentes de luz con emisiones en la banda azul. Por ello, para poder cuantificar esta emisiones en la banda azul, se creó un parámetro llamado **índice espectral G** que caracteriza las propiedades espectrales de las fuentes de luz, posibilitando su clasificación de modo cuantitativo y preciso en función de la cantidad real de luz azul emitida respecto al visible.

Una vez se ha explicado todo lo anterior, tenemos ya unos conceptos básicos sobre este campo, pero falta explicar qué es como tal la contaminación lumínica, siendo nuestro objeto de estudio.

Se considera como **contaminación lumínica**, al flujo lumínico que se proyecta hacia el cielo nocturno, siendo una combinación de la reflexión del suelo más el flujo que procede directamente de la luminaria. También se le puede llamar **FHSi** (Flujo Hemisférico Superior instalado). La contaminación lumínica es un porcentaje, del flujo lumínico de la propia luminaria y la cantidad de ese flujo lumínico que se dirige hacia el cielo nocturno.

Como la contaminación lumínica depende del lugar, existen distintos tipos de **E** que se utilizan para ver qué tipo de niveles de contaminación están permitidos en cada lugar. Una excepción por ejemplo es una carretera que permite unos niveles de contaminación lumínica mayores para asegurar la integridad vial.

Respecto al artículo 63 de la ley 7/2007, del 9 de julio, se establecen los siguientes tipo de áreas lumínicas:

- **E1** -> Áreas oscuras. Zonas de interés para la investigación científica (observación astronómica) y zonas no urbanizables incluidas en espacios naturales (con cierto régimen especial como convenios, normas internacionales, hábitats y especies de gran valor, etc)

- **E2** -> Áreas que admiten flujo luminoso reducido, como por ejemplo, terrenos urbanizables y terrenos no urbanizables que no sean de tipo 1.
- **E3** -> Áreas que admiten flujo luminoso intermedio, como por ejemplo, zonas industriales, zonas residenciales de interior de casco urbano y periferia donde la densidad de edificación es medio-baja, zonas que son utilizables en horario nocturno, sistema general de espacios libres.
- **E4** -> Áreas de flujo luminoso elevado. Zonas incluidas en el casco urbano con alta densidad de edificación y zonas de carácter comercial, turístico y recreativo de horario nocturno.

CLASIFICACIÓN DE ZONAS	DESCRIPCIÓN
E1	<b>ÁREAS CON ENTORNOS O PAISAJES OSCUROS:</b> Observatorios astronómicos de categoría internacional, parques nacionales, espacios de interés natural, áreas de protección especial (red natura, zonas de protección de aves, etc.), donde las carreteras están sin iluminar.
E2	<b>ÁREAS DE BRILLO O LUMINOSIDAD BAJA:</b> Zonas periurbanas o extrarradios de las ciudades, suelos no urbanizables, áreas rurales y sectores generalmente situados fuera de las áreas residenciales urbanas o industriales, donde las carreteras están iluminadas.
E3	<b>ÁREAS DE BRILLO O LUMINOSIDAD MEDIA:</b> Zonas urbanas residenciales, donde las calzadas (vías de tráfico rodado y aceras) están iluminadas.
E4	<b>ÁREAS DE BRILLO O LUMINOSIDAD ALTA:</b> Centros urbanos, zonas residenciales, sectores comerciales y de ocio, con elevada actividad durante la franja horaria nocturna.

Imagen 8.1.1: Imagen de la Junta de Andalucía. (ASPECTOS VIGENTES EN MATERIA DE PRESERVACIÓN DEL CIELO NOCTURNO EN ANDALUCÍA, TRAS LA ANULACIÓN DEL DECRETO, n.d.)

El porcentaje de cada tipo de E según la Junta de Andalucía es el siguiente:

- $E1 \leq 1\%$
- $E2 \leq 5\%$
- $E3 \leq 15\%$
- $E4 \leq 25\%$
- Para alumbrado viario general  $\leq 5\%$

Tras realizar un estudio de este campo y viendo toda la información obtenida, es necesario descartar ciertos parámetros posibles como pueden ser los equipos auxiliares que puede tener una luminaria (reloj astronómico, células fotovoltaicas, etc) ya que difieren en parte de nuestro objetivo que es la contaminación lumínica. Hay otros parámetros que son necesarios eliminarlos aunque sí pueden ser interesantes como puede ser los casos de estudio de lluvia, un coche debajo de la luminaria o el desgaste del pavimento por el tránsito, ya que puede cambiar la reflectancia del suelo, pero como el modelo va a trabajar en un entorno controlable como es un laboratorio, añadir estos parámetros carece de sentido aunque en un entorno real al estar orientado a luminaria exterior si pudieran ser útiles.

También para evitar que el modelo tenga excesivos parámetros, se van a eliminar parámetros que podrían ser interesantes tener en cuenta en un futuro, como puede ser el índice g o reflector de la luminaria, entre otros. Cabe destacar que tras observarse los temas de orientación y estructura de luminarias, estos son excesivos para abordarlos en este trabajo de fin de grado, por consiguiente han sido descartados. Se usarán en principio luminarias de tipo LED sin tener en cuenta su estructura u óptica.

Por consiguiente, y contando con la colaboración e implicación de expertos del dpto. de Ingeniería Eléctrica de la Universidad de Córdoba, los datos de interés que será necesario adquirir para el funcionamiento del sistema predictivo son los siguientes (en el siguiente apartado se explica el por qué de algunos y su obtención):

- Tipo de Luminaria
- Altura de la Luminaria (cm)
- Flujo Lumínico de la Luminaria (lm)
- Temperatura Correlada con el Color (K)
- Iluminancia en el Suelo (lux)
- Espectro de la luz (valor máximo nm)
- Color del Suelo
- Reflectancia del Suelo
- Iluminancia encima de la Luminaria (Lux)

Estos dos parámetros se obtienen para ver la contaminación lumínica y se extraen de los datos anteriores (no se usan al entrenar el modelo).

- Flujo Lumínico Superior a Luminaria (lm)
- FHSI (%)

En la siguiente tabla se pueden observar algunos ejemplos de patrones distintos:

	Ejemplo 1	Ejemplo 2	Ejemplo 3
<b>Tipo de Luminaria</b>	Philips Quijote BRP400 LED54-4s/740dm5 0	Philips CoreLine tempo medium BVP125 LED 67-4S/830	Philips Iridium Gen3 Mini BGP381 GRN45/740
<b>Altura de la Luminaria (cm)</b>	80	160	80
<b>Flujo Lumínico de la Luminaria (lm)</b>	5400	6724	4500
<b>Temperatura Correlada con el Color (K)</b>	4257	3181	4225
<b>Iluminancia en el Suelo (lux)</b>	1633	918	1939
<b>Espectro de la luz (valor máximo nm)</b>	444	605	450
<b>Color del Suelo</b>	negro	marrón oscuro	negro
<b>Reflectancia del Suelo</b>	4	12	4
<b>Iluminancia encima de la Luminaria (Lux)</b>	46.84	48.28	36.52
<b>Flujo Lumínico Superior a Luminaria (lm)</b>	154.89	353.63	84.76
<b>FHSI (%)</b>	2.87%	5.26%	1.88%

## 8.2. Adquisición de los datos

Para poder realizar la adquisición de los datos, se pensó en **realizar un experimento en un entorno controlado para simular la luminaria exterior**. Algo parecido a utilizar una luminaria de exterior y aislarla de todo tipo de iluminación, para que no se vieran los datos de la misma influenciados por otras luminarias. También se debía tener en cuenta que el suelo de la calle puede ser muy distinto, por ejemplo, desde césped hasta distintos tipos de cemento (dependiendo de la reflectancia R1, R2, R3 y R4).

Teniendo en cuenta el presupuesto y el tiempo disponible, aparte de tener en cuenta de que es un TFG, se llegó a la siguiente conclusión, para poder obtener los datos anteriormente nombrados.

Se pretendía crear una estructura con cierta altura que permitiera estar cerrada para impedir la entrada de luz y que en su interior se pudiera colocar la luminaria. Aparte, utilizar cartulinas para simular distintos suelos, de esta forma, si se disponen de los sensores adecuados, se podría simular en un entorno controlado la toma de datos de una luminaria de exterior.

Para realizar el experimento, se montó a mano la estructura. Está formada por cuatro barras de 240 cm, donde en la parte superior y en la parte inferior de la misma se colocaron dos planchas de metal con un metro cuadrado para permitir dar mayor estabilidad a la estructura. Aparte se pusieron a 80 cm y 160 cm dos refuerzos para dar mayor estabilidad a la estructura.

Cabe recalcar, que para la realización del montaje se tuvo que comprar guantes para evitar cortes con los hierros y del material necesario como tornillos o llaves inglesas.

En la siguiente imagen se puede ir viendo el progreso realizado, en concreto el montaje de la estructura sin colocar el techo y sin el refuerzo de 160 cm:



Imagen 8.2.1: Estructura de adquisición de datos en desarrollo

A la hora de montar la estructura se encontraron distintos problemas como que el tamaño de la estructura es prácticamente similar al techo del laboratorio, que los tornillos deben de ir hacia fuera para evitar problemas a la hora de atornillarlos para evitar dañar la tela o incluso añadir triángulos de refuerzo a la base para dar mayor estabilidad a la estructura y que no tuviera la posibilidad de colapsar. Todos estos problemas se fueron solucionando a medida que se terminaba de construir la estructura.

Para terminar la estructura, a la hora de terminar el techo, se aprovechó para atornillarlo con una tela mate para evitar utilizar pinzas en el techo y evitar el brillo de la estructura a la hora de realizar pruebas.

Una vez se tuvo la estructura se añadió una tela opaca colgando del techo con pinzas para evitar la entrada y salida de luz.

Se utilizaron pinzas para evitar que se cayera y para fijarla con toda la estructura. También se añadió una tela negra en la base para que a la hora de poner las cartulinas, si alguna dejaba un hueco, esta luz incidiera en la tela y no refleja como el metal. También se añadió un metro para que fácilmente se viera a qué altura estaba la lámpara u otras tomas de datos.

Finalmente se añadió la primera luminaria a la estructura teniendo ya una versión funcional para extraer datos. En estas imágenes se puede ver la estructura finalizada por dentro y por fuera con una luminaria encendida y con un ejemplo con cartulinas colocadas:

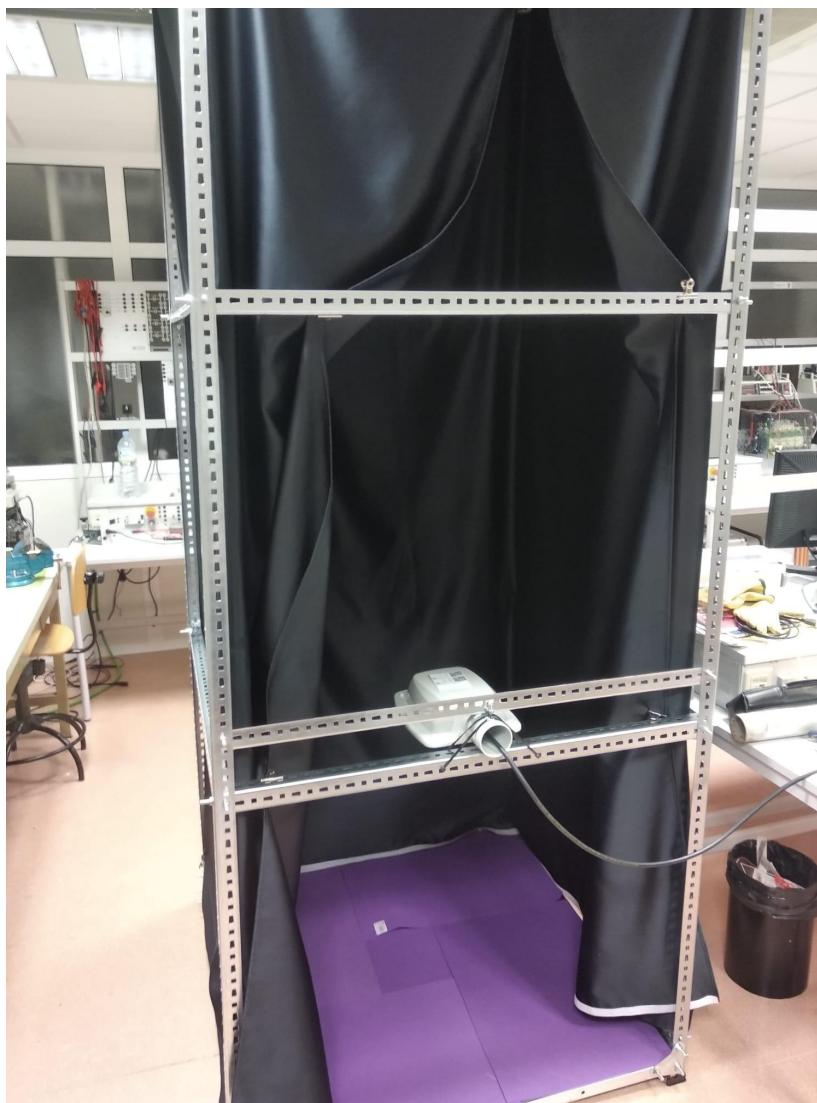


Imagen 8.2.2: Estructura de adquisición de datos en desarrollo con tela colocada

En la anterior imagen falta la tela del suelo y el metro, en la siguiente imagen si se puede ver la estructura completa.



Imagen 8.2.3: Estructura de adquisición de datos terminada



Imagen 8.2.4: Foto por encima de la luminaria



Imagen 8.2.5: Foto desde el suelo dentro de la estructura

Estas tres imágenes vistas, nos ofrecen una idea de cómo es la estructura finalizada por dentro, tanto desde abajo, en la luminaria y en el techo de la estructura.

Una vez realizada la estructura ya se pueden extraer patrones, pero cabe destacar que cada vez que se cambia la luminaria es necesario cambiar la estructura y todo lo que conlleva, no siendo un problema trivial.

Aparte de toda la estructura anterior se disponen de bridas para ayudar en el refuerzo de la luminaria en la estructura y de cartulinas de colores que permiten la simulación de distintos suelos, en concreto se disponía de:

- Negro
- Gris Oscuro
- Gris Claro
- Blanco
- Morado
- Rojo
- Verde Claro
- Verde
- Verde Oscuro
- Marrón Claro
- Marrón Oscuro

Esto permite simular distintos cementos con distintas reflectancias, distintos tipos de césped, o casos más extraños (como el color morado) que pueden ser interesantes para un caso de estudio como este.

Una vez tenemos todos los materiales, para comprender mejor cómo se extraen cada dato, se procede a explicar cómo se obtenía cada uno, teniendo en cuenta que son:

- Tipo de Luminaria
- Altura de la Luminaria (cm)
- Flujo Lumínico de la Luminaria (lm)
- Temperatura Correlada con el Color (K)
- Iluminancia en el Suelo (lux)
- Espectro de la luz (valor máximo nm)
- Color del Suelo
- Reflectancia del Suelo
- Iluminancia encima de la Luminaria (Lux)

### 8.2.1. Tipo de Luminaria

El tipo de luminaria es conocido, ya que es puesto a mano en la estructura. Estas luminarias han sido proporcionadas por el profesor Eduardo Ruiz Vela y modificadas por Álvaro David Domínguez López. Ya que estas luminarias necesitaban un cable para conectarlas a un enchufe para poder apagarlas y encenderlas.

Como es necesario en ciertos puntos saber más datos de las mismas (por ejemplo el flujo lumínico), se instaló una aplicación en el dispositivo móvil del autor del tfg llamada “Service Tag” que permite la obtención de más datos de la luminaria escaneada de philips. Esto es posible ya que las luminarias cuentan con un código QR que, gracias a esta aplicación, te ofrece toda la información relacionada con esa luminaria.

Las luminarias usadas son las siguientes:

Philips Quijote BRP400 LED54-4s/740dm50



Imagen 8.2.1.1: luminaria Philips Quijote BRP400 LED54-4s/740dm50

Usando la aplicación anteriormente comentada, nos ofrece los siguientes datos adicionales:

Quijote	
Propiedades	Ordenar A-Z
Designación	BRP400 LED54-4S/740 I DM50 DDF27
Flujo lumínico de la fuente de luz (lm)	5400
Color claro (K)	4000
Óptica	DM50
Número de LEDs	20
Consumo de energía CLO inicial (W)	37
Clase de seguridad eléctrica	I
Factor de potencia	0.9
Índice de rendimiento cromático	70
Temperatura ambiente (°C)	25
Tensión de alimentación (V)	220-240V
Frecuencia de suministro (Hz)	50/60Hz
Marca	Philips
 Producto	 Carro
 Log	 Más

Imagen 8.2.1.2: características luminaria Philips Quijote BRP400 LED54-4s/740dm50

Philips ClearWay BGP303 LED35-4S/740

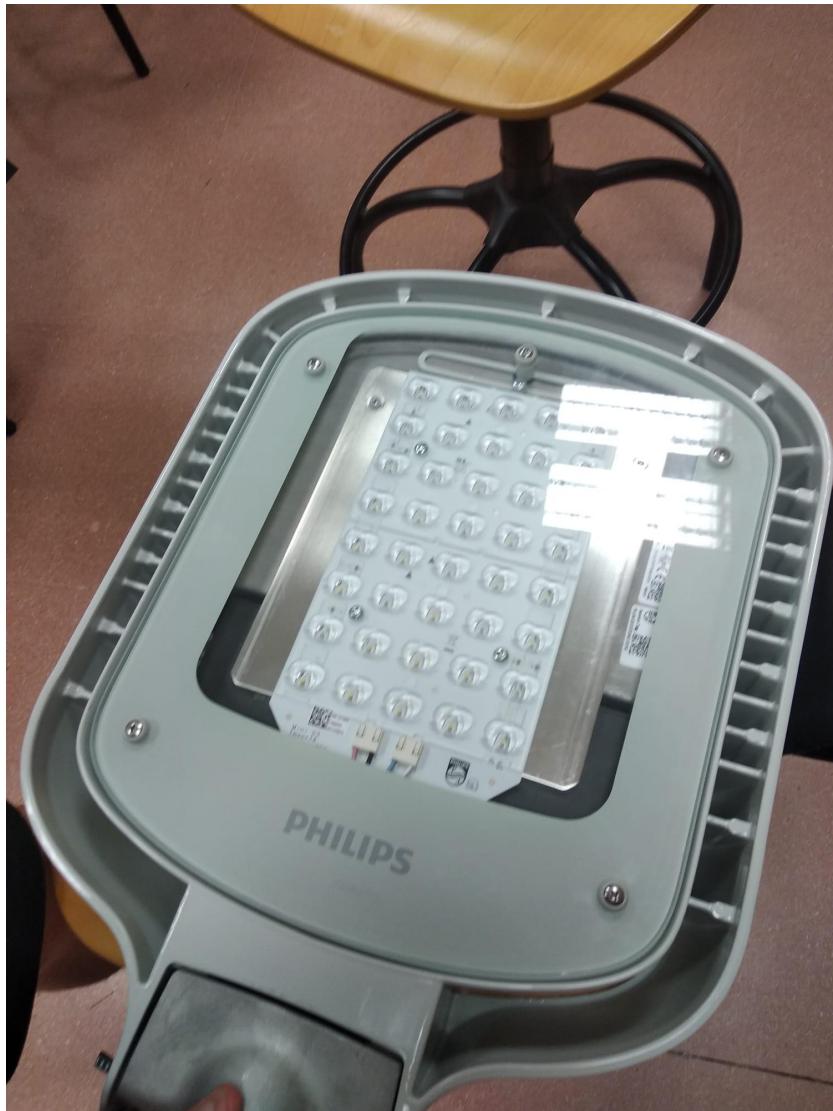


Imagen 8.2.1.3: luminaria Philips ClearWay BGP303 LED35-4S/740

Cabe destacar que esta luminaria es mucho más pesada que la anterior, de tal forma que se tuvo buscar una alternativa para colocarla en la estructura, ya que existía el riesgo de que se cayera o de que partiera el hierro de la estructura. Finalmente ideando un contrapeso improvisado y sujetando en parte la lámpara, se pudieron extraer los datos sin ocurrir ningún incidente.

Usando la aplicación anteriormente comentada, esta luminaria nos ofrece los siguientes datos adicionales:



Designación	BGP303 LED35-4S/740 PSR I DW10 DDF27 SRG10 42/60		
Flujo lumínico de la fuente de luz (lm)	3500		
Color de la luminaria	GR		
Color claro (K)	4000		
Óptica	DW10		
Número de LEDs	30		
Consumo de energía CLO inicial (W)	22.6		
Clase de seguridad eléctrica	I		
Factor de potencia	0.9		
Índice de rendimiento cromático	72		
Temperatura ambiente (°C)	35		
Tensión de alimentación (V)	~220V-240V		
Frecuencia de suministro (Hz)	50/60Hz		
Producto	Carro	Log	Más

Imagen 8.2.1.4: características luminaria Philips ClearWay BGP303 LED35-4S/740

Philips CoreLine tempo medium BVP125 LED 67-4S/830



Imagen 8.2.1.5: luminaria Philips CoreLine tempo medium BVP125 LED 67-4S/830

Esta luminaria tiene la curiosidad de que es la misma que se puede encontrar en el alumbrado exterior al edificio Leonardo Da Vinci de la UCO. Por lo tanto, si a posteriori se pretende realizar un prueba a nivel de campo, sería sencillo, ya que, se podrían extraer los datos en la azotea de dicho edificio.

Usando la aplicación anteriormente comentada, esta luminaria nos ofrece los siguientes datos adicionales:

12:07	...	4G	Wi-Fi	71
< CoreLine tempo medium				
Designación	BVP125 LED67-4S/830 PSU OFA52 ALU C1KC3			
Flujo del Sistema (lm)	6724			
Color claro (K)	3000			
Óptica	OFA52			
Número de LEDs	28			
Consumo de energía CLO inicial (W)	67			
Clase de seguridad eléctrica	I			
Factor de potencia	0.98			
Índice de rendimiento cromático	80			
Temperatura ambiente (°C)	35			
Tensión de alimentación (V)	220-240V			
Frecuencia de suministro (Hz)	50/60Hz			
Marca	Philips			
Producto	Carro	Log	Más	

Imagen 8.2.1.6: características luminaria Philips CoreLine tempo medium BVP125 LED 67-4S/830

Philips Iridium Gen3 Mini BGP381 GRN45/740



Imagen 8.2.1.7: luminaria Philips Iridium Gen3 Mini BGP381 GRN45/740

Esta luminaria, no tenía código QR, por tanto, con la aplicación “Service Tag” no se podían extraer los datos, investigando por internet y gracias a las siguiente pegatina incluida en la propia luminaria, se pudo extraer toda la información necesaria.



Imagen 8.2.1.8: pegatina luminaria Philips Iridium Gen3 Mini BGP381 GRN45/740

Hispaled Vera Series VRS60W R4S PF 3000K



Imagen 8.2.1.9: luminaria Hispaled Vera Series VRS60W R4S PF 3000K

Esta luminaria, no tenía código QR, por tanto, con la aplicación “ServiceTag” no se podían extraer los datos, investigando por internet y gracias a las siguiente pegatina incluida en la propia luminaria, se puedo extraer toda la información necesaria.



Imagen 8.2.1.10: pegatina luminaria Hispaled Vera Series VRS60W R4S PF 3000K

Philips EW BLAST POWERCORE CK Intelligent series



Imagen 8.2.1.11: luminaria Philips EW BLAST POWERCORE CK Intelligent series

Esta luminaria era muy antigua, no tenía código QR ni código manual para usar en la aplicación “Service Tag”. Se estuvo investigando por internet (con poco éxito), pero gracias a las diversas pegatinas incluidas en la propia luminaria, se pudo extraer toda la información necesaria. Son las siguientes:

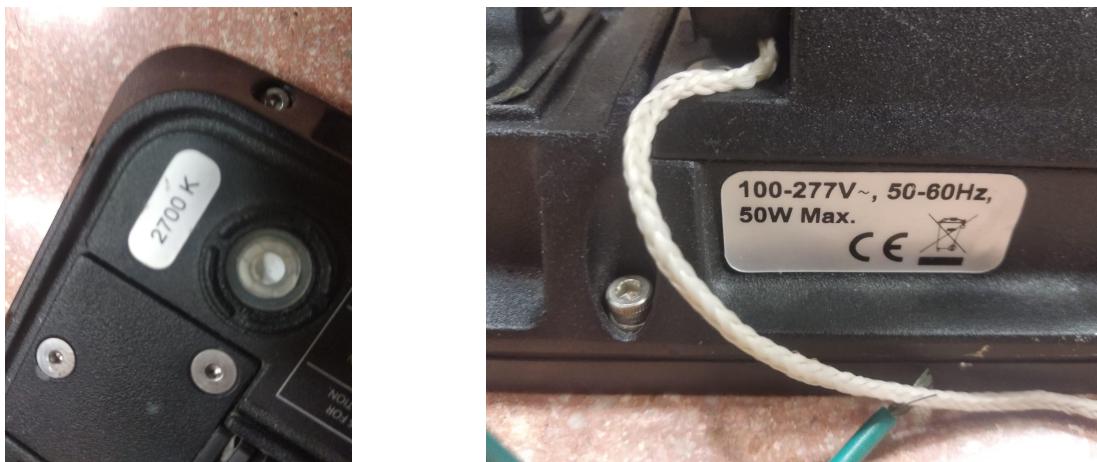


Imagen 8.2.1.12: pegatinas de la luminaria Philips EW BLAST POWERCORE CK Intelligent series

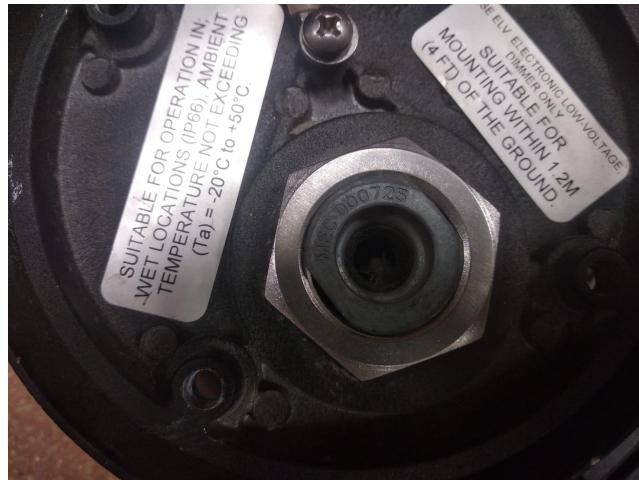


Imagen 8.2.1.13: pegatina de la base de la luminaria Philips EW BLAST POWERCORE CK Intelligent series

#### Philips RetroFit BGS981(Experimental)



Imagen 8.2.1.14: luminaria Philips RetroFit BGS981(Experimental)

Esta luminaria generó bastantes problemas, ya que está formada por distintas partes, es decir la óptica con las led es diferente al modelo del driver de la luminaria, aun así se decidió incluirla para obtener mayor variedad (como una luminaria experimental), la información necesaria se extrajo de las pegatinas que incluía la propia luminaria. Las imágenes son las siguientes:



Imagen 8.2.1.15: pegatina de la luminaria Philips RetroFit BGS981(Experimental)



Imagen 8.2.1.16: pegatina de la alimentación de la luminaria Philips RetroFit BGS981(Experimental)

Philips Coreline Malaga LED BRP102 LED55/740 II DM



Imagen 8.2.1.17: luminaria Philips Coreline Malaga LED BRP102 LED55/740 II DM

Esta luminaria era como las primeras vistas, bastante nueva, disponía de código QR, facilitando la toma de sus datos con la aplicación “ServiceTag”.

Usando la aplicación anteriormente comentada, nos ofrece los siguientes datos adicionales:

CoreLine Malaga LED	
Designación	BRP102 LED55/740 II DM
Flujo lumínico de la fuente de luz (lm)	4600
Color de la luminaria	GR
Color claro (K)	4000
Óptica	DM
Número de LEDs	60
Consumo de energía CLO inicial (W)	39
Clase de seguridad eléctrica	II
Factor de potencia	0.98
Índice de rendimiento cromático	70
Temperatura ambiente (°C)	35
Tensión de alimentación (V)	220 V;240 V
Frecuencia de suministro (Hz)	50/60Hz
Marca	Philips
 Producto	 Carro
 Log	 Más

Imagen 8.2.1.18: características luminaria Philips Coreline Malaga LED BRP102 LED55/740 II DM

### 8.2.2. Altura de la Luminaria

La extracción del atributo de la altura de la luminaria es muy sencillo de obtener ya que únicamente, es necesario una vez se encuentre instalada la luminaria, comprobar el metro colocado en la estructura.

Este dato se mide en **cm** y se añadió al patrón para comprobar si es un atributo interesante que afecte a la contaminación lumínica.

Para las pruebas llevadas a cabo, solo se disponen de dos tipos distintos de alturas, **80 y 160 cm**. Estas dos medidas se deben a que, respecto al tamaño total de la estructura, se decidió probar con dos medidas distintas. De tal forma que si el tamaño total de la estructura es 240 cm, se dividió en 0, 80, 160 y 240.

Como no se puede poner la luminaria en el suelo y en el techo se escogieron esas dos medidas.

### 8.2.3. Flujo Lumínico de la Luminaria

La extracción de este atributo también es relativamente sencilla. Únicamente es necesario ver la información extra de la luminaria instalada con la aplicación, y podemos encontrar un apartado que indica el flujo lumínico de la luminaria. En el caso de no contar con la aplicación, en la propia luminaria suele venir esta información.

Este dato se mide en **lúmenes** y es un dato muy importante ya que, el porcentaje de contaminación lumínica se basa en el flujo lumínico de la luminaria respecto al flujo que llega al cielo nocturno. Por tanto este dato es **indispensable** para el modelo.

#### 8.2.4. Temperatura Correlada con el Color

Respecto a este atributo, en el modelo se cree que no sería indispensable, pero para otros atributos como la reflectancia es indispensable como se verá posteriormente. Se considera como un atributo similar a la altura, ya que puede ser interesante y quizás afecta a la hora de crear el modelo.

Este atributo se mide en **K** y se puede medir gracias al **sensor uprtek premium**, este espectrógrafo ofrece mucha información como el espectro, luxes o la temperatura correlada con el color. Para obtener la **K** se realizó un disparo con el uprtek desde la parte más baja de la estructura apuntando a la luminaria, teniendo toda la estructura cerrada y aparte, apagando las luces del laboratorio para evitar luz intrusiva.

Cabe destacar, que cada vez que se va a usar el sensor uprtek es necesario hacer una calibración del negro, colocando la tapadera que tiene el propio sensor.

Una imagen del uprtek es la siguiente:



Imagen 8.2.4.1: imagen del sensor uprtek premium

En el apartado de básico se podía encontrar los **luxes** y la **k**, mientras que en el **espectro** encontramos el pico de energía del espectro.

### 8.2.5. Iluminancia en el Suelo

Para la iluminancia en el suelo se encontraron ciertos problemas. Se disponía de dos sensores, el **uprtek** y **Ix9621 philips**.

En la siguiente imagen se puede ver el luxómetro **Ix9621 philips** utilizado:



Imagen 8.2.5.1: imagen del sensor uprtek premium

Ambos daban resultados diferentes, tras realizar ciertas pruebas se llegó a la conclusión de que **Uprtek** era mucho más preciso, aparte era más sencillo de utilizar a la hora de realizar las pruebas.

Para tomar este dato, se utilizaba el mismo disparo realizado por uprtek cuando se realiza en el suelo, para obtener los lux que llegan hasta el suelo antes de reflejar. Este tipo de dato se mide en lux.

### 8.2.6. Espectro de la luz

Este atributo es similar, a la hora de adquirirlo, a la iluminancia en el suelo y a la temperatura correlada del color.

Tras realizar el disparo con el sensor **uprtek**, en vez de entrar en la pestaña de básico, entramos en la de “espectro” donde se nos muestra para cada longitud de onda la cantidad de energía que hay.

Para nuestro caso, para evitar coger todas las energías para cada tipo de onda, se va a escoger el valor máximo que marca **uprtek**, este valor se mide en nanómetros.

### 8.2.7. Color del Suelo

Al colocarse la cartulina de forma manual, ya se sabe el tipo de color que estamos usando para extraer ese patrón, por tanto este atributo es **tribial**. Los colores de cartulina que se disponen son:

- Negro
- Gris Oscuro
- Gris Claro
- Blanco
- Morado
- Rojo
- Verde Claro
- Verde
- Verde Oscuro
- Marrón Claro
- Marrón Oscuro

### 8.2.8. Reflectancia del Suelo

Este atributo no es trivial, de hecho, se encuentra relacionado con el **color** y con la **temperatura correlada del color (TCC)**. Para obtener la reflectancia se ha usado la siguiente tabla proporcionada por Ramón Lara, que se puede apreciar en la siguiente imagen:



Imagen 8.2.8.1: tabla de reflectancias de los colores dependiendo de la TCC

En esta tabla podemos comparar las cartulinas con los colores de la tabla obteniendo la reflectancia de cada una. Por eso es importante el **color**, pero aparte, la **temperatura correlada del color importa**, ya que si es un **color frío** la reflectancia escogida es el valor de la izquierda, si el color es un **color cálido**, la reflectancia es el valor de la derecha, y por último, si el color tiene un valor intermedio de la temperatura correlada del color, **se escoge la media de los dos valores**.

Este atributo se considera muy importante ya que influye en demasiados atributos como color, temperatura correlada del color, y también afectará a los lux que sobrepasan la luminaria por encima, haciendo que reboten más o menos rayos de luz en el suelo.

### 8.2.9. Iluminancia encima de la Luminaria

La iluminancia encima de la luminaria, es igual que cuando se mide en el suelo, es decir, se realiza un disparo con el sensor **uprtek** justo por encima de la luminaria apuntando hacia el suelo. De tal forma que se miden los lux que llegan a ese punto (donde se encuentra uprtek).

Este valor es el más importante, de hecho es el valor a predecir en el modelo, ya que este es el valor con el que sacamos el **flujo superior** y por tanto el **porcentaje de contaminación lumínica**. En el siguiente apartado se explica cómo se obtiene este porcentaje.

### 8.2.10. Flujo Lumínico Superior a Luminaria y FHSi

Estos datos se encuentran en el *calc*, pero no se usarán en el modelo. Estos datos son interesantes porque podemos ir viendo la contaminación lumínica de los patrones que vamos sacando.

Como los instrumentos para sacar el flujo lumínico **son muy costosos y complejos**, se decidió usar el siguiente método donde se usaba **uprtek** en el suelo y justo encima de la luminaria para extraer la iluminancia y obtener así el flujo superior. Una vez con el flujo superior, al tener los dos flujos, se puede extraer el **porcentaje de contaminación lumínica**.

Tras hablar con Eduardo Ruiz Vela, un especialista en el campo, nos comentó la siguiente solución para no usar instrumentos para sacar el flujo lumínico.

La iluminancia, como hemos visto anteriormente, es la cantidad de luz que llega a una superficie. A nivel de formula sería:

$$\text{Iluminancia} = \text{Flujo Luminoso} / \text{Área}$$

Si suponemos que tomamos todas las medidas en un punto, tenemos para la medida en el suelo que:

$$\text{Iluminancia Suelo} = \text{Flujo Total} / \text{Punto}$$

Siendo el Flujo total, el Flujo luminoso proporcionado por la luminaria y que a priori conocemos o que se puede conocer de forma sencilla por las especificaciones de la luminaria.

Por lo tanto, si aplicamos esta mism fórmula a la toma superior de la luminaria obtenemos que:

$$\text{Iluminancia Superior} = \text{Flujo Superior} / \text{Punto}$$

El datos que nos falta sería el flujo superior, pero como tenemos las dos siguiente ecuaciones:

$$\begin{aligned}\text{Iluminancia Suelo} &= \text{Flujo Total} / \text{Punto} \\ \text{Iluminancia Superior} &= \text{Flujo Superior} / \text{Punto}\end{aligned}$$

Cómo nos estamos basando en un punto y es lo mismo, este valor se puede eliminar quedando entonces:

$$\text{Flujo Total} / \text{Iluminancia Suelo} = \text{Flujo Superior} / \text{Iluminancia Superior}$$

Si sepáramos el Flujo Superior que es nuestro objetivo:

$$\text{Flujo Superior} = \text{Flujo Total} * (\text{Iluminancia Superior} / \text{Iluminancia Suelo})$$

De tal forma que todos los datos anteriores están disponibles. El flujo total, se puede extraer de la lámpara y las dos luminancias se pueden extraer con dos disparos realizados por uprtek, tanto en el suelo como justo encima de la luminaria.

Realizar distintos disparos por encima de la luminaria carece de sentido ya que se calcularía la variación de intensidad de la luz a medida que aumenta la distancia, eso difiere del flujo lumínico que es constante por encima de la luminaria y por tanto difiere del objetivo de ver si hay contaminación lumínica en esas condiciones.

Una vez se dispone de ambos flujos, con una regla de tres donde si el 100% es el flujo total, X será el porcentaje que se lanza al cielo nocturno. A nivel de fórmula sería:

$$X = (\text{Flujo Superior} / \text{Flujo Total}) * 100$$

Este % sería el valor a comparar con los distintos tipos de E para ver si esta luminaria estaría permitida en ese espacio.

Finalmente, una vez se extrajeron todos los patrones, se realizó una fotografía con todas las luminarias utilizadas juntas, siendo la siguiente:



Imagen 8.2.10.1: imagen de todas las iluminarias utilizadas en la adquisición de datos

Como podemos ver, tenemos **8 luminarias en total**, por cada una de ellas se extraen **11 patrones**, uno por cada tipo de cartulina simulando el suelo. Además, se miden en **2 alturas** distintas, para comprobar este parámetro. Finalmente para cada combinación se tomaban **2 iluminancias superiores**, para tener mayor cantidad de patrones y para evitar que si un dato es erróneo que afecte demasiado al modelo, al ser único.

Cabe destacar que la luminaria más pesada, Philips ClearWay BGP303 LED35-4S/740, no se pudo medir a mayor altura para mantener la integridad de la misma, ya que podría dañarse si se caía o partía la estructura, por su peso.

Por lo tanto, finalmente, se obtuvieron en la toma de patrones la cantidad de **330 patrones** para su uso en el modelo.

### 8.3. Preprocesamiento de los datos y conclusiones sobre la elección de características

Respecto a este apartado, es un punto un poco apartado del sistema de análisis y predicción de contaminación lumínica como tal, ya que, nos permite entender y comprender mejor los datos obtenidos de la extracción de datos y cómo se comportan entre sí, dichas características.

Primeramente, los datos obtenidos se encuentran en un fichero **DatosLuminaria.odt**. Ya que finalmente se usó calc para almacenar los datos, en la imagen siguiente se puede observar la estructura, en dicha imagen no se encuentran todos los datos pero en el **DatosLuminaria.odt** se pueden encontrar los 330 patrones:

	A	B	C	D	E	F	G	H	I	J	K
1	Tipo de Luminaria	Altura de la Luminaria(cm)	Flujo Luminico	Temperatura Luminaria	Especro d'Color Suelo	Reflectante	Illuminancia	Flujo superior	FHSI		
2	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 negro	4	46.84	154.89	2.87	
3	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 gris oscuro	20	74.35	245.86	4.55	
4	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 gris claro	42	118.00	390.20	7.23	
5	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 blanco	66	152.60	504.62	9.34	
6	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 morado	4	57.25	189.31	3.51	
7	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 rojo	10	50.37	166.56	3.08	
8	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 verde claro	55	152.20	503.29	9.32	
9	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 verde	27	129.80	429.22	7.95	
10	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 Verde oscuro	11	100.20	331.34	6.14	
11	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 marron claro	46	149.00	492.71	9.12	
12	Philips Quijote BRP400 LED54-4S/740dm50	80	5400	4257	1633	444 marron oscuro	11	65.00	214.94	3.98	
13	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 negro	4	27.16	367.58	6.81	
14	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 gris oscuro	20	31.48	426.05	7.89	
15	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 gris claro	42	45.19	611.59	11.33	
16	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 blanco	66	53.96	730.29	13.52	
17	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 morado	4	32.43	438.90	8.13	
18	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 rojo	10	32.43	438.90	8.13	
19	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 verde claro	55	46.35	627.29	11.62	
20	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 verde	27	39.60	535.94	9.92	
21	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 Verde oscuro	11	35.88	485.59	8.90	
22	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 marron claro	46	40.68	550.56	10.20	
23	Philips Quijote BRP400 LED54-4S/740dm50	160	5400	4257	399	444 marron oscuro	11	33.38	451.76	8.37	
24	Philips ClearWay BGP303 LED35-4S/740	80	3500	4144	1473	442 negro	4	35.56	44.45	1.27	
25	Philips ClearWay BGP303 LED35-4S/740	80	3500	4144	1473	442 gris oscuro	20	69.54	86.93	2.48	
26	Philips ClearWay BGP303 LED35-4S/740	80	3500	4144	1473	442 gris claro	42	123.00	153.75	4.39	
27	Philips ClearWay BGP303 LED35-4S/740	80	3500	4144	1473	442 blanco	66	207.50	259.38	7.41	
28	Philips ClearWay BGP303 LED35-4S/740	80	3500	4144	1473	442 morado	4	51.76	64.70	1.85	
29	Philips ClearWay BGP303 LED35-4S/740	80	3500	4144	1473	442 rojo	10	54.47	68.09	1.95	

Imagen 8.3.1: calc con los patrones obtenidos en la adquisición de los datos

Estos datos para poder ser utilizados, se necesita una transformación a un fichero .csv. Pero antes se realizó una revisión de los datos para encontrar posibles fallos, como celdas con fórmulas erróneas que en pasos posteriores podían dar problemas.

Para transformar este .odt a un .csv es necesario darle a la opción **guardar como** .csv y escoger el formato que queremos de separación de datos y saltos. Una vez hecho este paso, se corrigió un problema que había, ya que los números (los decimales en concreto), deben estar tras un punto y en el fichero odt estaban tras

una coma. Una vez solucionados estos problemas, el fichero debería de quedar con la siguiente forma, el nombre de este fichero es **DatosLuminaria.csv**:

1	TipoLuminaria,AlturaLuminaria,FlujoLuminicoTotal,TCC,IluminanciaAbajo,Espectro,ColorSuelo,ReflectanciaSuelo,IluminanciaSuperior,Flujo Superior,FHSI
2	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,negro,4,46.84,154.89,2.87
3	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,gris oscuro,20,74.35,245.86,4.55
4	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,gris claro,42,118.00,390,20,7.23
5	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,blanco,66,152.60,504.62,9.34
6	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,morado,4,57.25,189.31,3.51
7	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,rojo,10,50,37,166.56,3.08
8	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,verde claro,55,152.20,503.29,9.32
9	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,verde ,27,129,80,429,22,7.95
10	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,Verde oscuro,11,100.20,331.34,6.14
11	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,marrón claro,46,149.00,492.71,9.12
12	Philips Quijote BRP400 LED54-4s/740dm50,80,5400,4257,1633,444,marrón oscuro,11,65.00,214.94,3.98
13	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,negro,4,27.16,367.58,6.81
14	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,gris oscuro,20,31.48,426.05,7.89
15	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,gris claro,42,45.19,611.59,11.33
16	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,blanco,66,53.96,730.29,13.52
17	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,morado,4,32.43,438.90,8.13
18	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,rojo,10,32.43,438.90,8.13
19	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,verde claro,55,46.35,627.29,11.62
20	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,verde ,27,39.60,535.94,9.92
21	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,Verde oscuro,11,35.88,485.59,8.99
22	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,marrón claro,46,40.68,550.56,10.20
23	Philips Quijote BRP400 LED54-4s/740dm50,160,5400,4257,399,444,marrón oscuro,11,33.38,451.76,8.37
24	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,negro,4,35.56,44.45,1.27
25	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,gris oscuro,20,69.54,86.93,2.48
26	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,gris claro,42,123.00,153.75,4.39
27	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,blanco,66,207.50,259.38,7.41
28	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,morado,4,51.76,64.70,1.85
29	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,rojo,10,54.47,68.09,1.95
30	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,verde claro,55,146.30,182.88,5.23
31	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,verde ,27,127.40,159.23,4.55
32	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,Verde oscuro,11,67.33,84.16,2.40
33	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,marrón claro,46,133.30,166.63,4.76
34	Philips ClearWay BGP303 LED35-4s/740,80,3500,4144,1473,442,marrón oscuro,11,47.35,59.19,1.69
35	Philips CoreLine tempo medium BVP125 LED 67-4s/830,80,6724,3181,4329,605,negro,4,67.82,84.78,1.26
36	Philips CoreLine tempo medium BVP125 LED 67-4s/830,80,6724,3181,4329,605,gris oscuro,20,153.60,192.00,2.86
37	Philips CoreLine tempo medium BVP125 LED 67-4s/830,80,6724,3181,4329,605,gris claro,42,286.00,357.50,5.32

Imagen 8.3.2: datos con formato csv y listos para su uso

Una vez tenemos los datos en formato csv, se puede trabajar cómodamente con ellos. Para tener los datos recogidos en el mismo punto y por si en un futuro se tienen que añadir datos, se pretenden almacenar los datos en una base de datos.

Por ello, se creó una base de datos de **mysql** llamada **db** con la tabla **datos\_luminaria**, con todas las características que se tomaron en la fase de adquisición de los datos. Es una tabla muy sencilla, que su objetivo es almacenar los datos como si de un fichero se tratase, da igual que haya patrones repetidos en el caso de que los haya, ya que puede darse el caso de haber tomado dos muestras idénticas. Una imagen de la base de datos es la siguiente:

Field	Type	Null	Key	Default	Extra
TipoLuminaria	varchar(255)	YES		NULL	
AlturaLuminaria	float	YES		NULL	
FlujoLuminicoTotal	float	YES		NULL	
TCC	float	YES		NULL	
IluminanciaAbajo	float	YES		NULL	
Especetro	float	YES		NULL	
ColorSuelo	varchar(255)	YES		NULL	
ReflectanciaSuelo	float	YES		NULL	
IluminanciaSuperior	float	YES		NULL	
FlujoSuperior	float	YES		NULL	
FHSI	float	YES		NULL	

Imagen 8.3.3: show que muestra los atributos de la tabla datos\_luminaria

Como la creación del servicio mysql se iba a utilizar también en el sistema, se aprovechó para integrarlo en docker, igual que la inserción de datos a la base de datos. Se aprovechó para, en la aplicación donde se encuentra el modelo, crear la inserción de datos. Estos puntos se explicarán mejor en la última fase.

Una vez los datos ya se encuentran almacenados en la base de datos, se creó el fichero de python **preprocesamiento.py** (que va aparte del docker) donde se realizaron las pruebas necesarias para extraer información interesante de los datos almacenados. Cabe destacar que como va a parte del docker, es necesario instalarse localmente los paquetes necesarios en **preprocesamiento.py**. Aparte de tener el servicio de la base de datos activo.

En este programa se pretende obtener unos histogramas que muestran información de la distribución de los datos y sobre todo, un mapa de correlaciones que nos permita ver las relaciones entre atributos y los atributos con el atributo objetivo.

Solo se extraerán de la base de datos aquellos datos que son interesantes para su estudio (todos menos Flujo Superior y FHSI). Aparte, se realizarán pruebas con los datos normalizados y estandarizados.

Respecto a porqué se realiza la **estandarización** y **normalización** de los datos, se debe a que la diferencia de rangos puede hacer que unos datos opaquen a otros, es decir, si por ejemplo, midieramos la distancia a nivel de km para un parámetro y distancias a nivel de cm para otro parámetros no serían comparables, en el sentido

de que se le daría mayor prioridad a los km ya que tienen mayor valor y un rango mucho más amplio.

Por eso es necesario que todos los datos que se vayan a usar en un modelo se encuentren normalizados o estandarizados para que su uso sea más sencillo (a la hora de entender los datos) y para evitar los problemas comentados anteriormente.

Cuando nos referimos a **normalización**, nos referimos a que los datos deben encontrarse escalado entre 0 y 1, esto se puede conseguir con la siguiente fórmula:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Respecto a la **estandarización**, se consigue con la siguiente fórmula:

$$\frac{X - \mu}{\sigma}$$

Tanto la normalización como la estandarización se puede realizar de forma muy sencilla con scikit, ya que posee funciones que permiten implementar estas fórmulas, siendo :

- **sklearn.preprocessing.StandardScaler**.
- **sklearn.preprocessing.MinMaxScaler**.

Cabe destacar que tanto **TipoLuminaria** como **ColorSuelo**, son dos tipos de características **categóricas**, por tanto las operaciones de rango y distancias entre valores carece de sentido, por eso es importante no darle valores numéricos. Por ejemplo, si rojo es igual a 1 y negro igual a 3, existe una distancia de 2 que no significa nada.

Por consiguiente, se va a usar **OneHotEncoder** de scikit que nos va a permitir modificar las columnas que queremos (en este caso hay que reorganizar el dataframe para que las dos columnas nominales están en los primeros lugares) para que si se pueda trabajar con esos datos.

Un ejemplo de su funcionamiento sería el siguiente:

Si tenemos 3 colores, rojo, amarillo y verde. La función, lo que haría sería que, rojo fuera 1,0,0; amarillo fuera 0,1,0 y verde 0,0,1.

De tal forma que se encuentran en distintas dimensiones y no hay distancias entre ellos, es decir, se consideraría que cada uno de ellos es un nuevo atributo que existe o no existe. Por lo tanto, si existe rojo, no existe amarillo ni verde.

Una vez aplicado todo lo visto anteriormente se lanzó el programa obteniendo los siguientes resultados, se dividen en dos imágenes para mayor claridad:

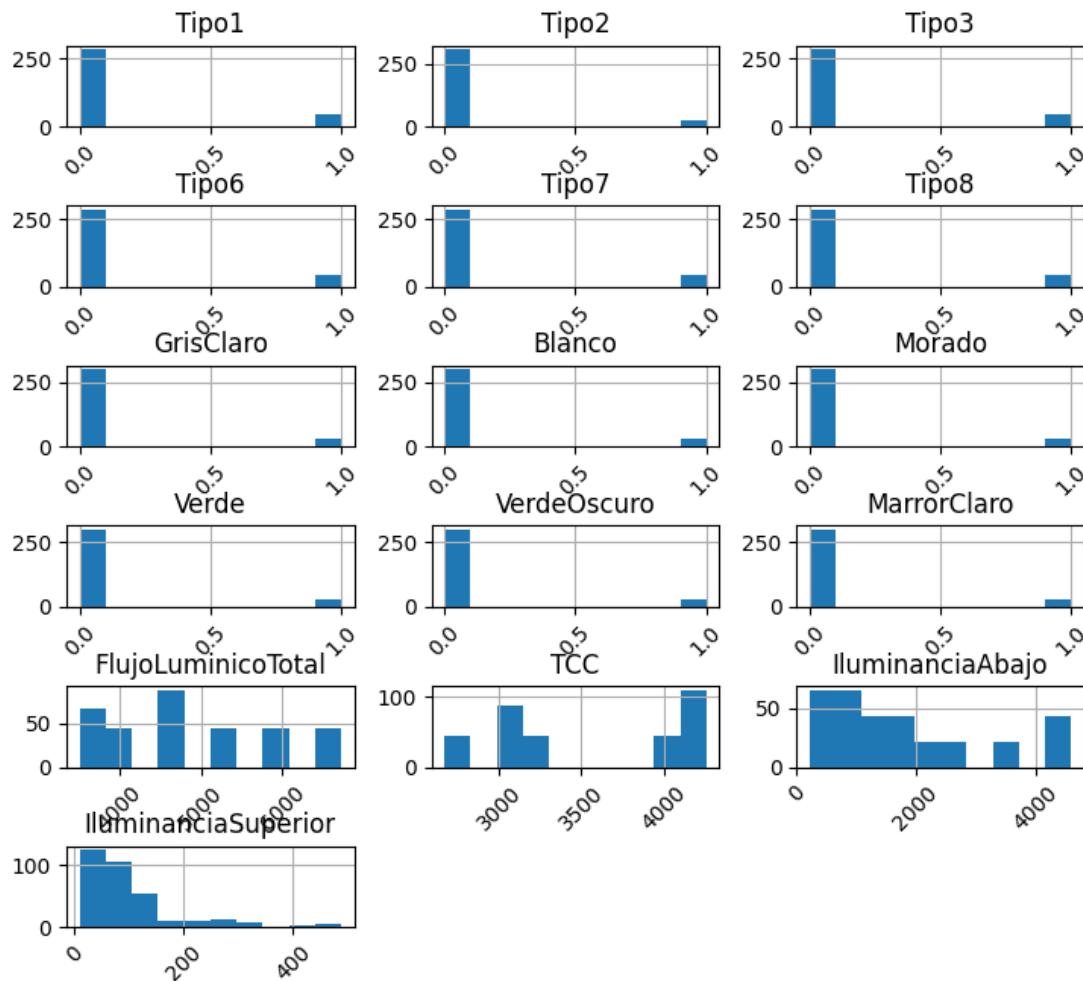


Imagen 8.3.4: histograma sin normalización o estandarización (primera parte)

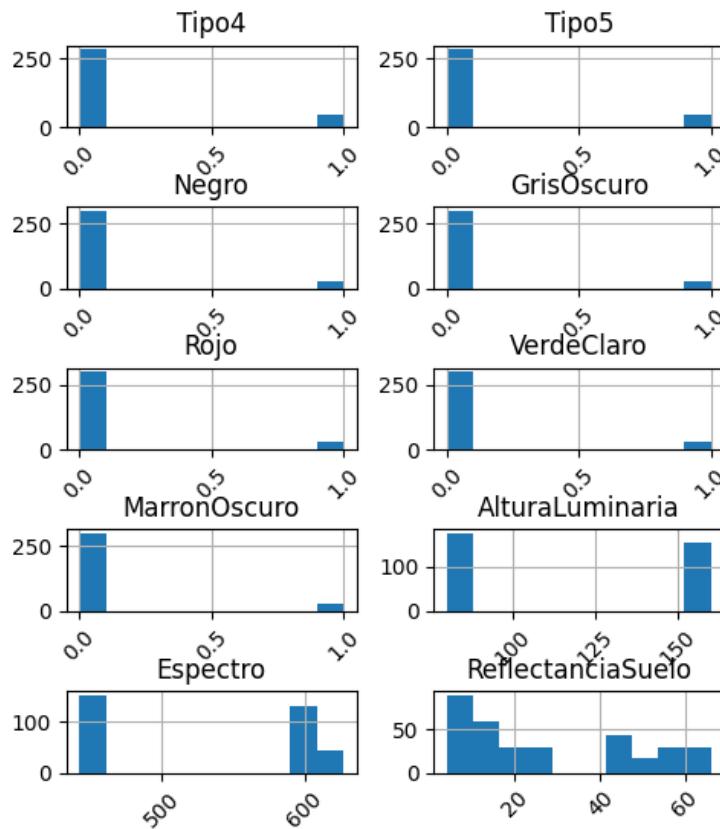


Imagen 8.3.5: histograma sin normalización o estandarización (segunda parte)

De estos histogramas se puede obtener bastante información. Cuando nos referimos al tipo 1, 2, 3, etc. Nos referimos al tipo de la luminaria que se ha descompuesto en 8 tipos distintos, lo mismo sucede con el color de suelo que se ha dividido en los 11 colores usados.

Los valores que antes eran categóricos, en el histograma poseen la misma estructura. Esto se debe a lo que se explicó anteriormente, 0 significa no existir y 1 existir, por tanto, hay 22 patrones que si existen de un tipo de luminaria 1 por ejemplo y 308 que no, esto sucede con todos los tipos de luminarias. Lo mismo sucede con los colores pero con diferente cantidad de patrones ya que por ejemplo rojo se repite con cada luminaria de tal forma que si existían dos patrones con características similares y dos alturas son 32 patrones, si quitamos dos casos de la luminaria que pesaba demasiado y no se puso a 160cm, entonces 30 patrones de rojo y 300 patrones que no son rojo. Esto sucede con cada patrón de color, tiene 30 patrones y 300 que no son de su tipo.

Respecto al resto de histogramas, lo que se debe de buscar es que se encuentre estratificado, es decir, que todos los valores tengan una cantidad equitativa de patrones para que el modelo pueda entrenar en las mejores condiciones. Más o menos se cumple en todos los histogramas, pero hay algunos que no se cumple del todo. Respecto a la altura si son la misma cantidad de patrones a 160 y 80 cm pero solo son esos valores ya que no se realizaron pruebas a otro tipo de alturas. Respecto al espectro, ocurre algo similar, como las luminarias eran cálidas o frías no obtuvimos un espectro con valores intermedios, aun así la distancia entre los valores no es muy excesiva. Por el resto de valores, están más o menos equilibrados.

Para la **normalización y estandarización**, las conclusiones son las mismas ya que los histogramas son idénticos, aun así se muestran los histogramas de cada uno respectivamente:

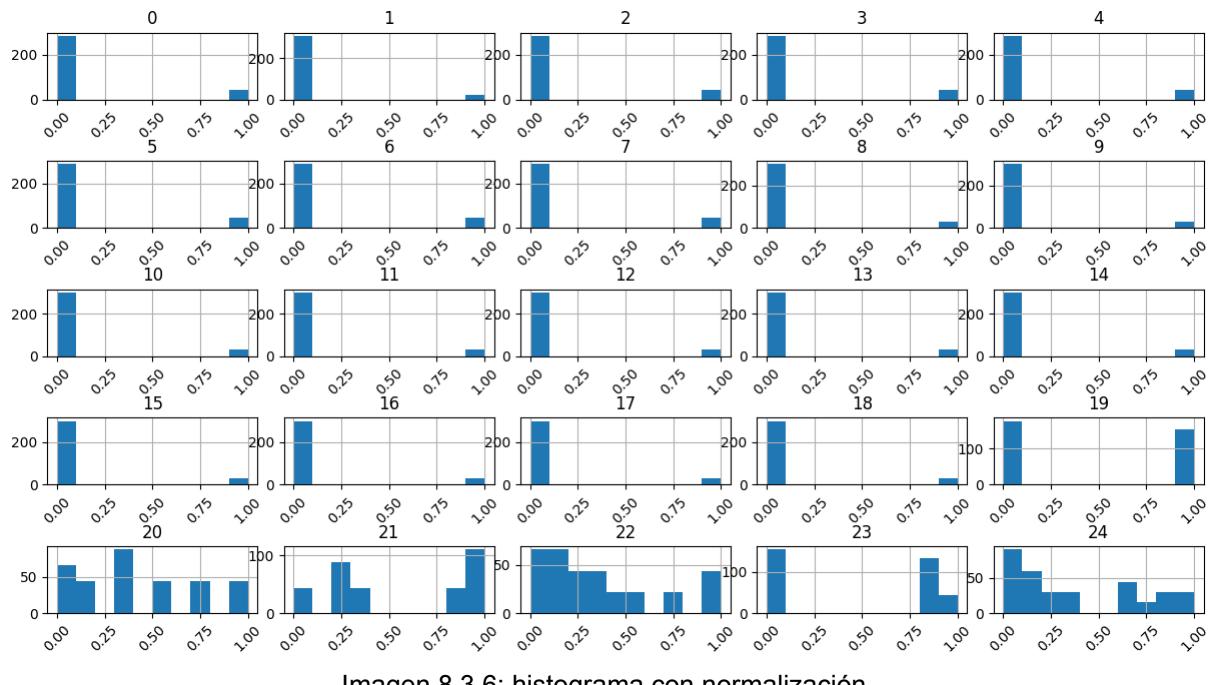


Imagen 8.3.6: histograma con normalización

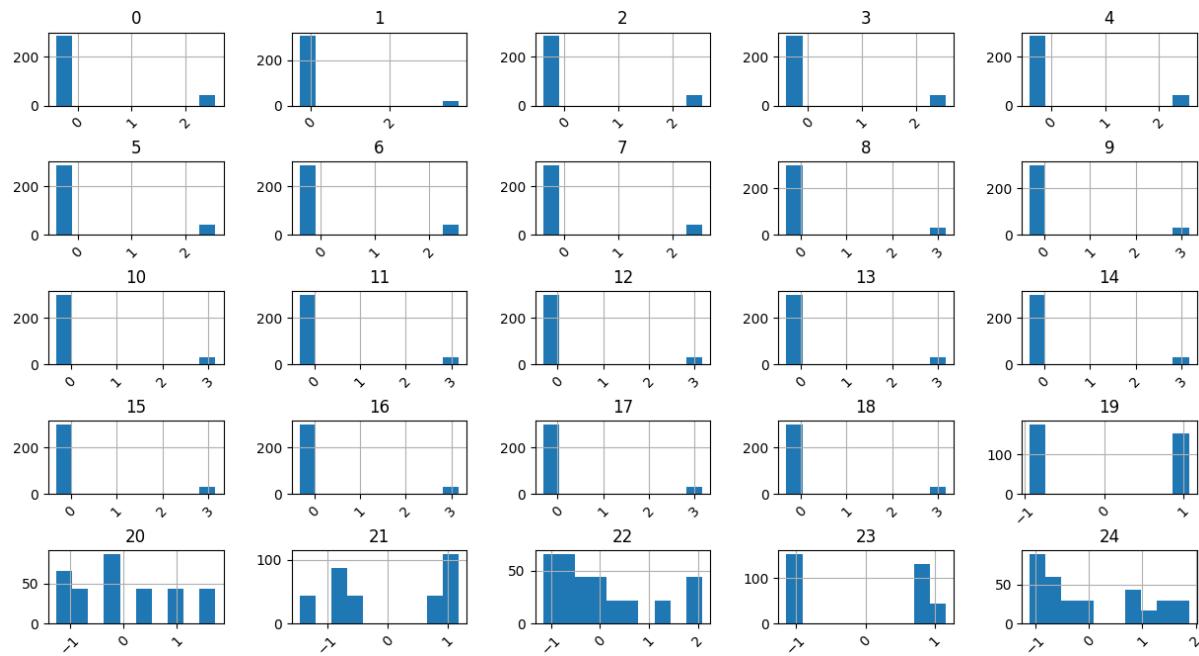


Imagen 8.3.7: histograma con estandarización

Para finalizar, se obtuvo el mapa de correlaciones siendo el siguiente:

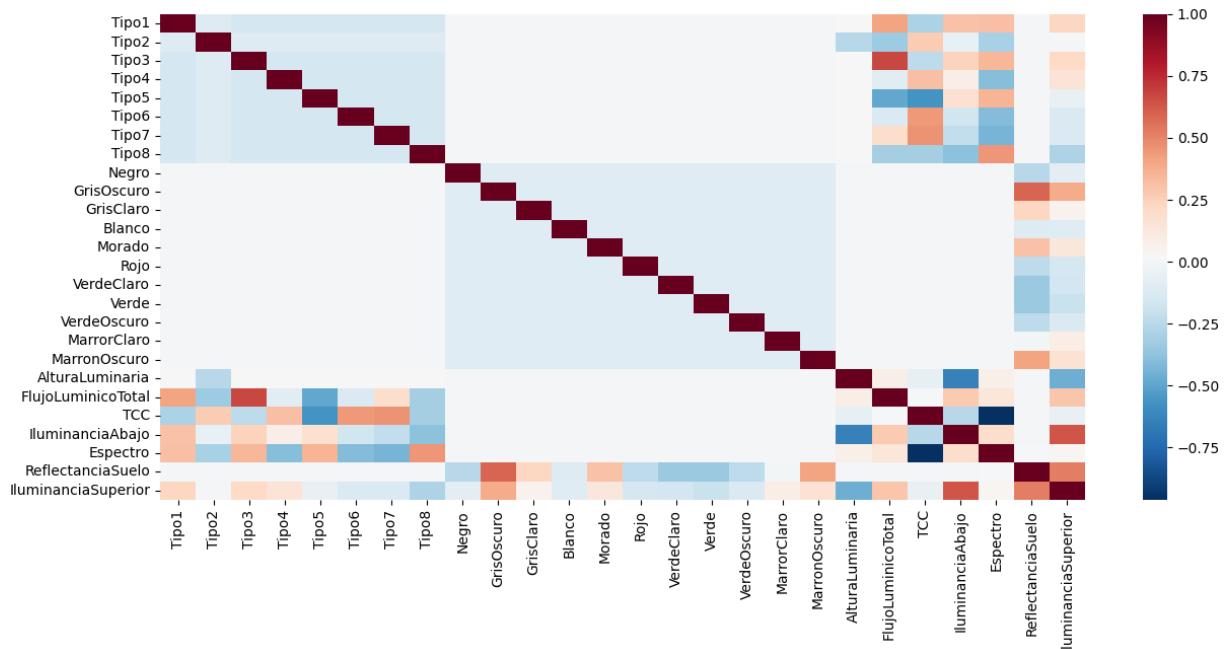


Imagen 8.3.8: mapa de correlaciones

De la imagen anterior se puede sacar muchísima información. Por ejemplo, todos los valores que se aproximen al valor 0, significa que están muy poco correlados, es decir que son indiferentes, si sus valores se aproximan a 1 o -1 significan que están muy correlados. Teniendo en cuenta siempre el atributo que estamos trabajando, si dos atributos están correlados, significa que uno es prescindible, y que se puede eliminar, y si un atributo está muy correlado con el atributo a predecir, significa que es imprescindible (a grandes rasgos, hay casos donde un atributo es imprescindible y se puede eliminar si se encuentra muy correlado con otro atributo imprescindible por ejemplo).

Si nos centramos en la imagen obtenida, podemos encontrar bloques muy grandes de blancos entre colores y luminarias y esto es lógico ya que son completamente independientes entre sí. Tanto colores entre sí como luminarias entre sí están poco correlación aunque un poco más que en el caso anterior.

También nos podemos encontrar que los colores del suelo son independientes a los flujo total, iluminancia abajo, espectro y TCC, esto es lógico ya que son valores que dependen más de la luminaria y de hecho, se encuentran algo correladas con las mismas.

Buscando casos más interesantes, encontramos que el TCC está muy fuertemente correlacionado con el espectro, como era de esperar, ya que el mayor valor de energía del espectro influye en el color de la luminaria, y por tanto, si luz fría o cálida.

Otro caso bastante interesante es la altura de la luminaria con la iluminancia superior e inferior. Este caso no era esperable pero a su vez es lógico que se encuentren correlacionados, ya que a mayor altura mayor contaminación vamos a tener y mayor iluminancia superior. Esto se debe a que la tela refleja algo de la luz, por lo tanto no se puede escapar horizontalmente (como si de una calle se tratase) viéndose obligada a ascender, aparte, cuando la luminaria está a poca altura, la propia luminaria puede reflejar la luz evitando que ésta acabe por encima de la misma. Por estos dos casos, se puede comprender que este relacionado de tal forma que mayor altura de la luminaria, mayor iluminancia superior (para este experimento, puede ser que a nivel de campo, no ocurra así)

Cabe destacar que la iluminancia inferior y superior están correlacionadas, esto era esperable ya que mayor iluminancia abajo implica mayor iluminancia arriba, ya que es proporcional.

Si llega más de lux abajo, mayor cantidad de luxes se reflejarán en el suelo y mayor cantidad de luxes sobrepasan la luminaria aumentando la iluminancia arriba.

Si nos centramos exclusivamente en la iluminancia superior, que es el valor a predecir para obtener la contaminación lumínica, nos encontramos que:

**Están correlados:**

- Reflectancia del suelo
- Iluminancia abajo
- Altura Luminaria
- Gris Oscuro
- Tipo 8

**Poco correlados:**

- Espectro
- TCC
- Algunos Colores
- Algunas Luminarias

Es lógico que teniendo tantos colores y tantas luminarias, puede ser que unas estén más correladas con la iluminación superior que otras, pero es curioso que el gris oscuro esté tan correlacionado, puede ser un caso excepcional por estos datos, quizás con una cantidad de patrones mayor no exista tanta correlación entre estos dos parámetros.

Respecto a los parámetros poco correlacionados, encontramos el TCC y el Espectro, estos parámetros entre sí, están muy correlacionados y, respecto a la iluminancia superior, muy poco, por tanto estos dos atributos podrían ser eliminados ya que no influyen prácticamente nada en la contaminación lumínica.

Esto es muy interesante ya que nos genera ciertas preguntas como, si no afecta el TCC ni el espectro, ¿el **índice espectral G** se podría descartar también?. Es una muy buena pregunta para investigar y descubrir en proyectos futuros.

Aun así, estos atributos se van a mantener a la hora de entrenar el modelo, ya que al ser una cantidad reducida de patrones no van a implicar demasiado tiempo de código y añadir algo de ruido puede permitir al modelo una mayor flexibilidad y evitar así que solo dependa de uno o dos atributos como mucho.

Únicamente se va a eliminar a la hora de entrenar el modelo el tipo de luminaria, ya que el nombre nos da igual, únicamente es interesante las características de la misma, como su flujo lumínico. También en un futuro se podrían añadir otras características de la luminaria como cantidad de LED o tipo de reflector, en el caso de que puedan ser interesantes aunque como vimos en la imagen de las correlaciones, no parece que el tipo de luminaria esté demasiado correlado con la iluminancia superior (extrayendo el caso del flujo y iluminancia producida por la luminaria).

## 8.4. Creación del sistema de análisis y predicción de contaminación lumínica

### 8.4.1. Tipos de modelos

Antes de comenzar a crear el sistema de análisis y predicción de contaminación lumínica, es necesario adentrarnos en el campo del aprendizaje automático y ver que tipo de modelos más importantes existen.

En la actualidad existen diferentes tipos de aprendizaje automático, existiendo el **aprendizaje supervisado y el aprendizaje no supervisado**. En nuestro caso, se usará el aprendizaje supervisado ya que se basa en los **problemas de clasificación y problemas de regresión** (Pal et al., 2016, #).

El aprendizaje no supervisado se basa en el **agrupamiento o clustering**, de tal forma que, los datos se introducen pero no sabemos el valor de salida, ya que no es una etiqueta o un valor numérico. Por lo tanto, el objetivo de este método de aprendizaje es descubrir en los patrones de entrada características, regularidades, correlaciones, comportamientos, categorías, etc.

#### 8.4.1.1. Problemas de clasificación.

Los **problemas de clasificación** aparecen cuando se pretende prever a qué tipo clase pertenece un patrón dentro de un conjunto de dos clases o más. Un problema de dos clases podría ser un problema donde a partir de ciertos datos con ciertas características, el modelo debe predecir si es una persona casada o no, por ejemplo.

También existen problemas multiclase, estos se encuentran dentro de este tipo de problemas, un ejemplo sería una foto de una letra donde el modelo a partir de esa foto (una vez esté ya entrenado) sea capaz de encontrar a qué clase de letra pertenece.

En resumen, los modelos de clasificación se utilizan cuando el resultado a obtener es una etiqueta discreta. Lo que quiere decir que, el resultado se encuentra en un conjunto finito de resultados.

#### 8.4.1.2. Problemas de regresión.

Por otra parte, existen problemas que simplemente no se pueden solucionar con sí o no, o incluso asignando a un tipo dentro de varias clases. Por ello, existen otro tipo de problemas denominados de regresión. Los problemas de regresión son aquellos que buscan dar un valor aproximado para predecir qué valor tomará ese dato con ciertas características, es decir, no busca clasificar el dato.

Un ejemplo de este tipo de problemas sería, intentar averiguar qué precio tendrá un refresco dentro de una semana, basándose en los datos obtenidos hasta ahora.

En resumen, los modelos que se basan en problemas de regresión buscan encontrar un método que relaciona un cierto conjunto de características (de las entradas al modelo) con una variable continua siendo el objetivo a conseguir.

#### 8.4.2. Modelos elegidos para usar en el sistema

Una vez comprendido la existencia de los dos tipos de problemas donde se busca clasificar un patrón o obtener un valor futuro a partir de los datos obtenidos, se puede comprender ante a qué tipo de problema nos encontramos. Como se ha ido explicando durante este proyecto, se busca a partir de ciertas características obtener la contaminación lumínica, y como se explicó en la fase de elección de características y adquisición de los datos, concretamente la iluminación superior.

Si se intenta buscar una relación entre este problema y los explicados anteriormente, se puede observar que no es un problema de clasificación, ya que no tenemos 2 o más clases donde se puede clasificar la iluminancia superior.

Más bien es un valor que se busca predecir a partir de ciertos datos, por tanto, nos encontramos ante un problema de regresión.

Una vez sabemos que **nuestro problema es de regresión**, es necesario buscar modelos que se asignen a estos tipos de problemas. Como los modelos principalmente se ven afectados por los datos con los que trabajan, se van a usar distintos tipos de modelos para ver cual de ellos ofrece mejores resultados. Hay gran variedad de modelos, pero para este caso se van a utilizar los siguientes:

##### 8.4.2.1. SVR:

Cuando se habla de SVM nos referimos a **máquinas de vectores soporte**, estas son muy utilizadas en problemas de clasificación, ya que se busca dividir dos o más clases a partir de un vector que se va ajustando teniendo en cuenta distintas dimensiones. Como no se puede utilizar SVM en este problema, se optó por SVR (Regresión de Vector Soporte). La principal diferencia entre estos modelos es que en SVM se busca minimizar la tasa de error mientras que en SVR se busca ajustar el error entre un umbral.

Además, este modelo se usó con anterioridad en la universidad por el autor del TFG, como se comprende en gran medida su funcionamiento, se sabe que es un buen modelo para problemas de clasificación, como en scikit existe un método donde se aplica SVR (scikit-learn developers (BSD License), 2007) que es similar al visto en la universidad, se decidió probar su funcionamiento para ver cómo se comporta ante estos datos.

#### 8.4.2.2. Árbol de decisión:

El **árbol de decisión**, de forma resumida, se basa principalmente en la toma de decisiones para ciertos atributos, llegando a predecir un valor. En cada nodo se toma una decisión para ir avanzando al siguiente nodo hasta llegar a un nodo hoja donde finalmente se llega al valor predecido.

A grandes rasgos este sería su funcionamiento, al recibir un patrón, desgrana los valores de cada atributo a medida que lo necesita descendiendo por el árbol hasta llegar a un valor. La forma en la que se crea dicho árbol es un poco compleja y no se entrará en mucho detalle, pero se basa en la búsqueda de la menor entropía (la entropía es la cantidad de desorden de la información, o la cantidad de aleatoriedad en los datos) de cada atributo para ir creando cada nodo.

Este tipo de modelo también se usó con anterioridad en la universidad previamente, aparte, en un pequeña investigación se llegó a la conclusión de que es un modelo muy utilizado para este tipo de problemas donde suele dar buenos resultados, por todo lo anterior se decidió probarlo para este problema.

Para este caso también se usará un método de scikit para su implementación (scikit-learn developers (BSD License), 2007)

#### 8.4.2.3. RandomForest:

Este modelo, a diferencia de los otros modelos vistos, es un **ensemble de modelos**, es decir, un modelo creado a partir de modelos. Donde, en este caso, está formado por una gran cantidad gran cantidad de árboles de decisión (el modelo anteriormente visto) para predecir un valor. Como finalmente se utilizó el árbol de decisión como modelo, se decidió ver si con uno solo árbol de decisión es suficiente para tener un resultado decente o si un **RandomForest**, al ser una mayor cantidad de árboles de decisión, proporcionaba mejores resultados. También se quiere comprobar el caso contrario, si por el hecho de usar un **RandomForest**, al añadir demasiados árboles decae la precisión del valor predicho.

También para este modelo se usará un método de scikit para su implementación (scikit-learn developers (BSD License), 2007)

#### 8.4.2.4. Regresión Lineal y SDGRegressor:

Principalmente se buscó utilizar un modelo de **Regresión lineal**, ya que estos son muy sencillos y, en muchos casos, para estos tipos de problemas pueden resultar útiles. El modelo de **Regresión lineal** es sencilla su implementación porque simplemente establece una recta para proporcionar la tendencia de un conjunto de datos, permitiendo predecir para cada patrón un valor de esta recta.

Aun así, se decidió buscar un tipo de Regresión Lineal distinto, llegando al **SDGRegressor**. Este modelo se basa en el gradiente descendente estocástico donde el gradiente de la pérdida se estima en todos los patrones a la vez, y el modelo se va actualizando con un programa de fuerza decreciente.

Este último modelo es bastante difícil de comprender pero se ha escogido para tener dos modelos parecidos basados en regresión lineal. Estos modelos suelen dar buenos resultados en problemas de predicción de precios, es decir donde el valor resultante depende mucho de valores de días u horas anteriores de manera que suelen ser constantes, aun así se han escogido para ver si en nuestro problema dan también buenos resultados.

Para ambos modelos se usaron implementaciones de scikit, para la **Regresión Lineal** (scikit-learn developers (BSD License), 2007) y para el **SDGRegressor** (scikit-learn developers (BSD License), 2007)

### 8.4.3. Tipos de errores escogidos

Una vez explicados todos los modelos, es necesario explicar cómo se pueden comparar, para ver cuánto son mejores unos frente a otros para los datos de nuestro problema.

Para ello, se basó principalmente en el MSE, aunque también se añadió el MAE para comprobar qué resultados nos iba dando. Se buscaba el MSE más bajo (tanto para entrenarse los modelos como en la comparativa entre distintos modelos). El MAE es únicamente informativo, no tiene impacto dentro del sistema.

#### 8.4.3.1. MSE

El **MSE** es el error cuadrático medio y nos da como resultado cuán bueno es un modelo, siendo más bajo peor. El objetivo es medir el error cuadrado promedio de nuestras predicciones. Para cada patrón, se calcula la diferencia cuadrada entre las predicciones y el objetivo. Posteriormente se promedian estos valores. Se podría resumir como el “error acumulado” de las predicciones.

#### 8.4.3.2. MAE

El **MAE** es el error absoluto medio, a diferencia del **MSE** no penaliza enormemente los patrones que son demasiado distintos del valor a predecir. Esto permite que no sea tan sensible a patrones con valores atípicos. Este se puede describir como el “error medio individual” de cada predicción.

#### 8.4.4. Planteamiento y creación del sistema

Una vez explicado el tipo de problema, los diferentes modelos que se van a utilizar y la comparativa de los mismos, solo faltaría explicar el diseño que se pretende utilizar para llevar a cabo el sistema. Cabe destacar que el sistema como tal será el conjunto de servicios que se encuentran en el docker, permitiendo su fácil instalación en cualquier dispositivo, aún así se han creado programas auxiliares muy interesantes para realizar preprocesamiento, un ejemplo de un pequeño programa que usaría un cliente para acceder al sistema y un programa generador de gráficas que ofrece información relevante sobre el sistema y sus resultados.

Para que se comprenda mejor la estructura general que se pretende llevar a cabo, se ha realizado el siguiente diagrama de casos de uso sobre el sistema completo:

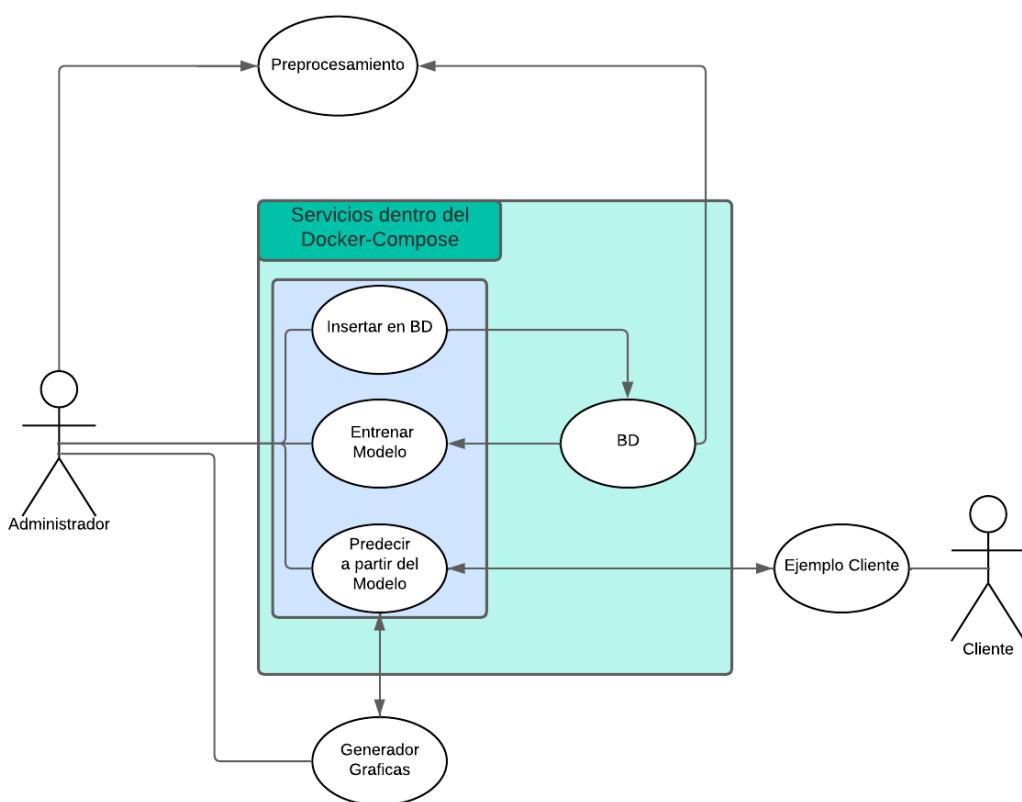


Imagen 8.4.4.1: diagrama del sistema completo

Como podemos ver en la imagen previa, nos encontramos dos tipos posibles de usuarios, el **administrador** y el **cliente**.

El **cliente** como tal, únicamente tendrá acceso a la api para que dado un patrón prediga el nivel de contaminación lumínica, para ello se ha creado un programa intermedio sencillo, para que pueda introducir dicho patrón. El cliente no tendrá acceso a nada relacionado al entrenamiento de modelos o inserciones en la bd de patrones. Se excluyen también los programas auxiliares creados para recabar información del funcionamiento del modelo y que se encuentran fuera del docker-compose.

El **administrador**, tendrá pleno control de todo el sistema, pudiendo usar sus programas auxiliares o utilizar cualquier llamada a la api, ya sea insertar, entrenar o predecir. Aparte, puede acceder a la base de datos y modificar todo aquello que sea necesario.

Una vez explicados todos los posibles personajes, se explicarán todos los casos de uso, entrando en mayor profundidad sobre su uso y funcionamiento.

Docker es una herramienta muy potente sobre creación de imágenes y contenedores permitiendo gran modularidad, por ello, se comenzó con la creación de un docker-compose que permite la inicialización de distintos servicios, en nuestro caso, se crearon dos servicios.

Para poder empezar con el proyecto se necesitan que los datos que se encuentran en un csv, se encuentren almacenados en una base de datos. Por tanto, se creó el primer servicio que coincide con el caso de uso BD.

#### 8.4.4.1. Servicios del sistema

En este apartado se explicaran los dos servicios que nos podemos encontrar dentro del docker compose como se vio en la imagen Imagen 8.4.4.1. Los dos servicios son **mysql** y **modelo\_prediccion\_contaminacion\_luminica**.

##### 8.4.4.1.1 mysql

El primer servicio es mysql. Este caso de uso constituye la base de datos del sistema, es el lugar donde, de forma sencilla, se almacenarán los datos del csv creado con anterioridad y donde se encuentran los patrones conseguidos en la fase de adquisición de los datos. Esto permitirá que si en un futuro se desea eliminar un atributo, simplemente no se escoja al realizarse un `select` a la base de datos. La base de datos se encuentra alojada en el puerto 3306, tanto del contenedor, como de la propia máquina donde se instale. Es una base de datos de mysql donde podremos encontrar un usuario con contraseña `password` y `root` con contraseña `password`. El nombre de la base de datos es `db` y únicamente albergará una tabla llamada **datos\_luminaria** que contendrá las características necesarias para poder albergar los datos.

##### 8.4.4.1.2 modelo\_prediccion\_contaminacion\_luminica

El otro servicio creado es **modelo\_prediccion\_contaminacion\_luminica**, este servicio pretende contener todo lo relacionado con el modelo, en concreto los casos de uso de **insertar en BD**, **entrenar modelo** y **predecir a partir de modelo**. Este servicio será una api, que dependiendo de la llamada se le realice, realizará dichos casos de uso. El puerto donde se albergará es el 8000, tanto en el contenedor como en la máquina donde se instale (estos puertos son fácilmente modificables dentro del docker-compose por si existiera algún otro servicio usando dicho puerto en la máquina donde se vaya a alojar el sistema). Este servicio tiene una dependencia, y es que este servicio no funcionará si no se inicia el servicio de la base de datos. Toda esta información se puede encontrar dentro del fichero docker-compose.yaml

En el siguiente diagrama de flujo se puede observar mejor cómo se encuentra dividido el servicio **modelo\_prediccion\_contaminacion\_luminica**.

**modelo\_prediccion\_contaminacion\_luminica**

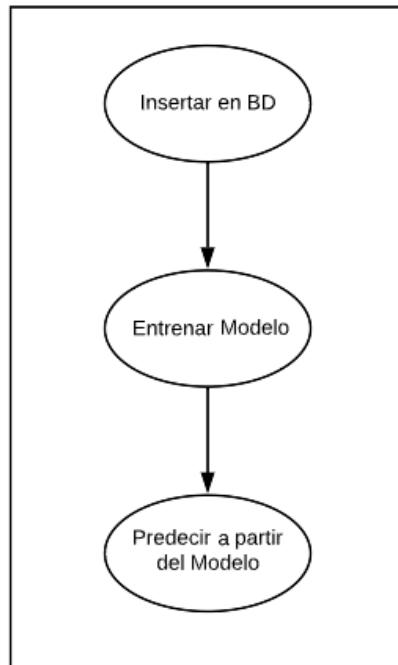


Imagen 8.4.4.1.2.1: diagrama de modelo\_prediccion\_contaminacion\_luminica

Cabe destacar que el orden de uso sería el que se puede apreciar en el diagrama, por lo tanto, las llamadas get al servicio serían en ese orden. Eso no implica que internamente Insertar llame a Entrenar Modelo, por ejemplo.

Tras haber explicado el caso de uso de bd y la existencia de dos servicios en el docker-compose, se explicarán los casos de uso que podemos encontrar dentro del servicio de **modelo\_prediccion\_contaminacion\_luminica**. Desglosando cada caso de uso para un mayor entendimiento.

#### 8.4.4.1.2.1 Insertar en BD

Como su propio nombre indica, el objetivo es insertar el fichero csv con patrones en la base de datos. Como este servicio es una api se explicará cómo se debe realizar la llamada get, ya que si realizamos una llamada get vacía, no se realizará ninguna acción.

El nombre de la petición es **insertarPatrones** y los parámetros de entrada que tiene son ambos obligatorios, siendo **base\_de\_datos** de tipo str y **archivo\_datos** de tipo str. Un ejemplo de llamada para su mayor entendimiento, sería el siguiente si se realiza desde navegador:

[http://localhost:8000/insertarPatrones/?base\\_de\\_datos=db&archivo\\_datos=DatosLuminaria.csv](http://localhost:8000/insertarPatrones/?base_de_datos=db&archivo_datos=DatosLuminaria.csv)

Donde la base de datos pasada es el nombre de nuestra base de datos y el archivo de datos el el nombre del fichero csv que queremos que se inserte en la bd.

Esta llamada no devuelve nada como tal pero avisa si la inserción es un éxito o ha ocurrido algún problema.

Para mayor entendimiento, se ha creado el siguiente diagrama de flujo:

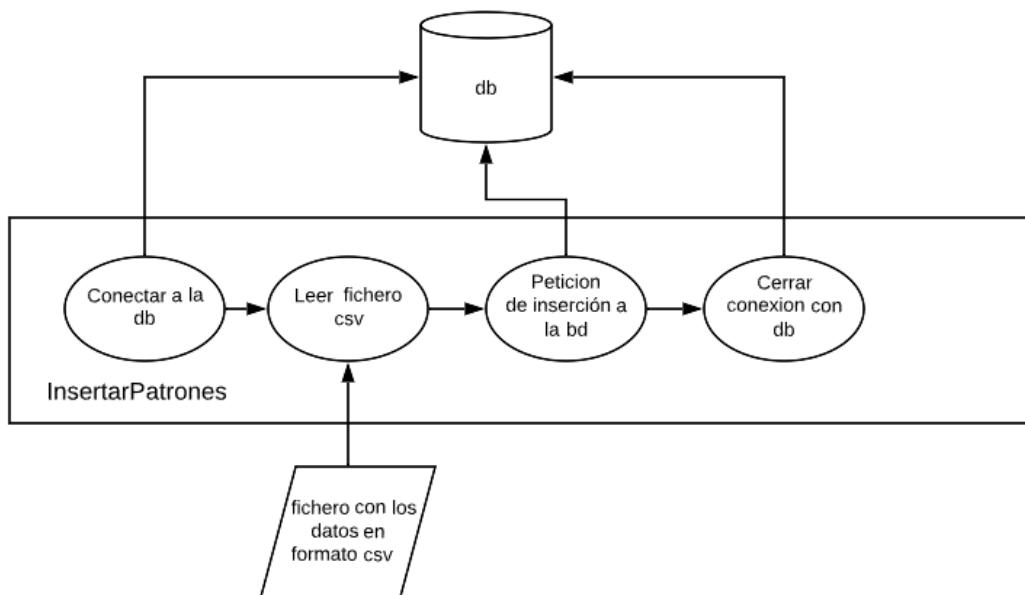


Imagen 8.4.4.1.2.1.1: diagrama de Insertar en BD

#### 8.4.4.1.2.2 Entrenar

Este caso de uso es algo más complejo, ya que entrenar modelo implica realizar distintas acciones. Antes de comentar como funciona se va a explicar las entradas y salidas. El nombre de la petición es **entrenar** y los parámetros de entrada que tiene son **base\_de\_datos** de tipo str y siendo obligatorio y **crear\_nuevo\_fichero\_entrenamiento** de tipo bool siendo opcional y por defecto True. Un ejemplo de llamada para su mayor entendimiento, sería el siguiente si se realiza desde navegador:

[http://localhost:8000/entrenar/?base\\_de\\_datos=db&crear\\_nuevo\\_fichero\\_entrenamiento=True](http://localhost:8000/entrenar/?base_de_datos=db&crear_nuevo_fichero_entrenamiento=True)

Donde la base de datos pasada es el nombre de nuestra base de datos y la flag determinará si se quiere crear un nuevo fichero de entrenamiento (normalmente si queremos que la flag esté a True pero hay un caso específico cuando se usa el generador de gráficas que es necesario que se encuentre a False, se explicara en dicho apartado el por qué).

Esta llamada no devuelve nada pero avisa si el entrenamiento es un éxito o ha ocurrido algún problema. Aparte, si el entrenamiento es un éxito, genera los siguientes archivos:



Imagen 8.4.4.1.2.2.1: ficheros generados tras el entrenamiento

Los archivos **.pickle** almacenan los modelos y los scalers. Los nombres de los modelos, almacenan los resultados obtenidos para cada combinación de patrones incluyendo el mejor caso.

Una vez se ha explicado su uso y lo que genera se explicará brevemente cómo funciona.

Primeramente, crea el fichero csv para entrenar a partir de la bd por eso se realiza una conexión (si la flag está desactivada no se haría este proceso). El uso de la flag solo se utiliza para el programa auxiliar de creación de gráficas, permitiendo que no se actualicen los patrones del csv y pudiendo así extraer ciertos patrones para que no se utilicen en el entrenamiento.

Una vez se tienen dichos datos, se lee el fichero y se pasa a dataset.

Cuando se obtiene el dataset se van llamando a los distintos tipos de modelos para que se entrene con cada tipo de modelo y dentro de cada tipo de modelo, con diferentes características para ver qué modelo ofrece un **MSE** menor. A cada entrenamiento se le pasan los patrones de test para que cuando entrene, pueda ver cuán bueno es el **MSE** para dicha combinación de parámetros. Una vez terminado el proceso de entrenamiento con todos los tipos de modelos y distintas características se comprueba cual tiene un menor **MSE** y el que de menor resultado se guarda en un pickle. Mientras se entrena se van guardando los mejores modelos para no perderlos, ya que, el modelo que ha dado mejor **MSE** con ciertas características, puede darse el caso de que no vuelva a dar el mismo **MSE** (por factores aleatorios).

Por tanto se van guardando los mejores y el mejor de todos se carga y finalmente se guarda como el mejor. Una vez guardado ya se habría realizado el entrenamiento por completo.

Para mayor entendimiento, se ha creado el siguiente diagrama de flujo:

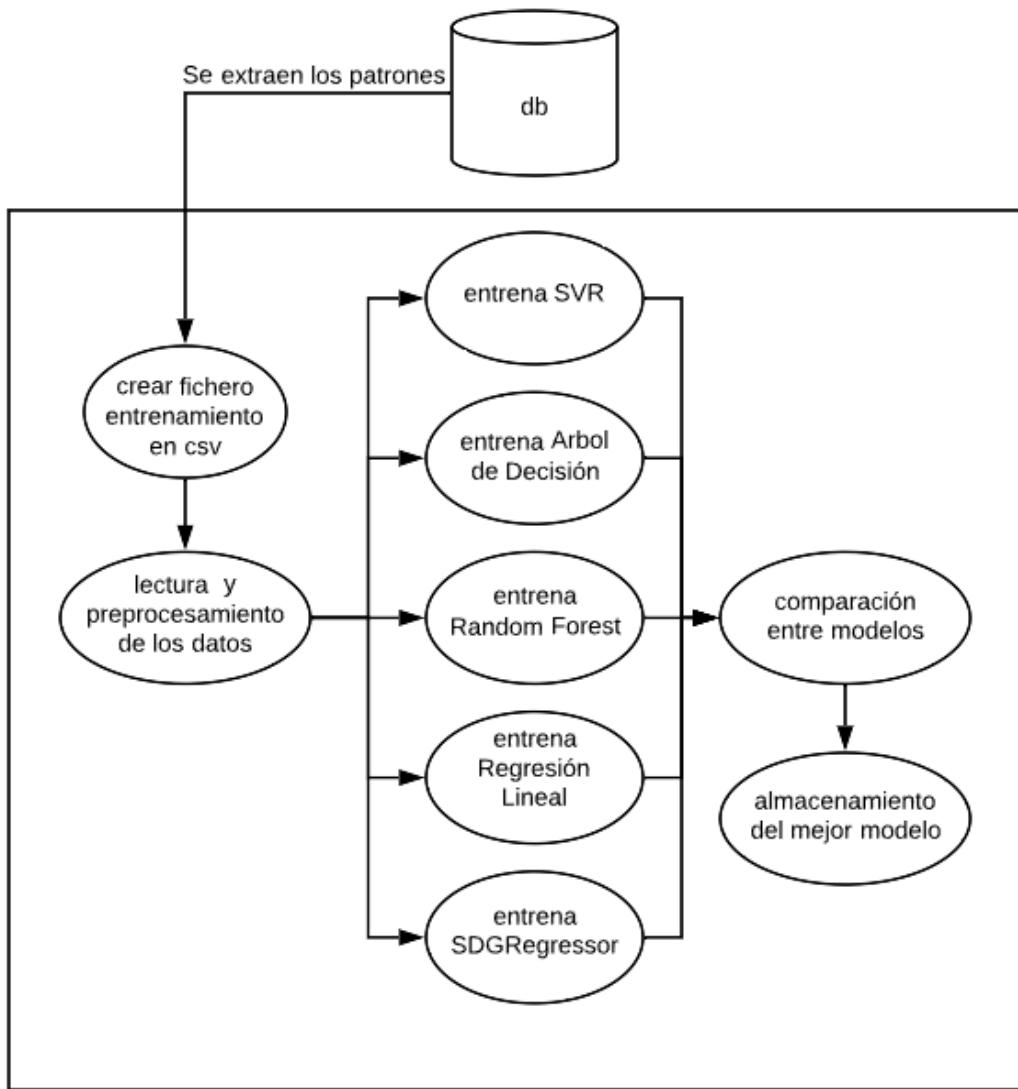


Imagen 8.4.4.1.2.2.2: diagrama del entrenamiento

Tras ver el diagrama de entrenamiento, se puede comprender mucho más fácilmente el funcionamiento del mismo, aun así se van a hacer los diagramas de cada entrenamiento del modelo ya que, a parte de entrenar, se guarda el modelo para permitir que cuando se comparan todos los modelos se escoja el de mayor **MSE**, permitiendo su carga para ya finalmente almacenar el mejor modelo.

Esto se realiza de esta forma ya que, si al reentrenar el mismo modelo con los mismos parámetros, dicho modelo puede dar diferentes **MSE** por ciertos factores aleatorios.

Los diagramas de flujo para cada tipo de entrenamiento son los siguientes:

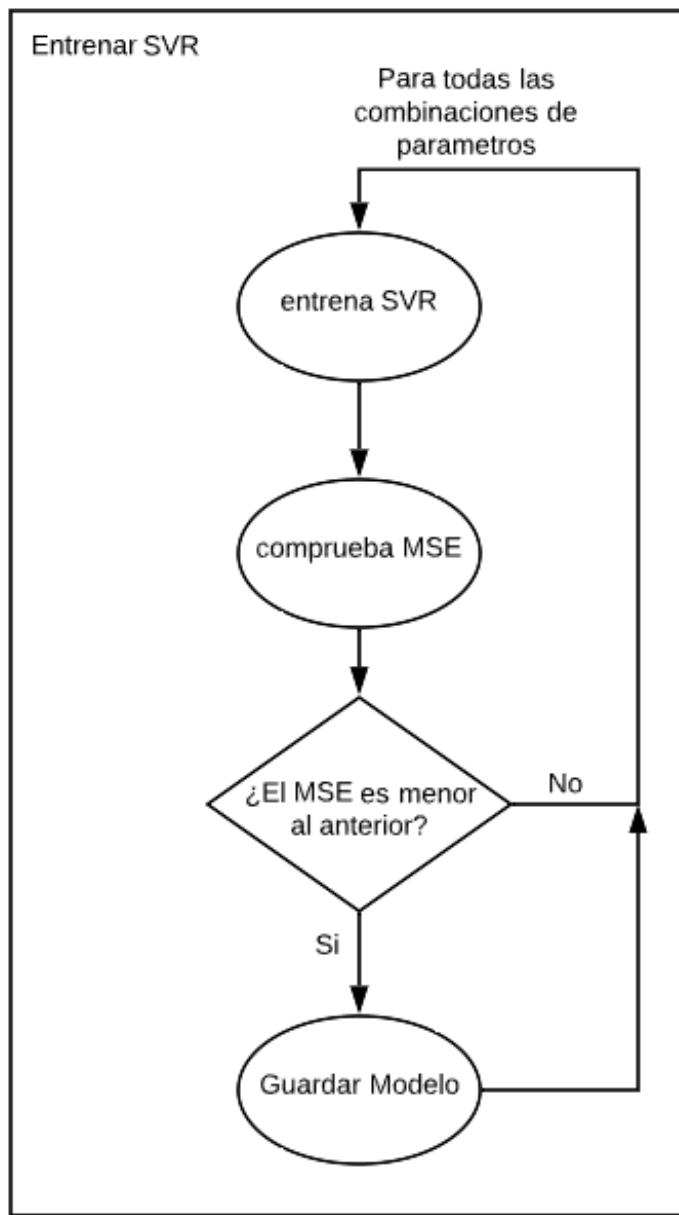


Imagen 8.4.4.1.2.2.3: diagrama entrenar SVR

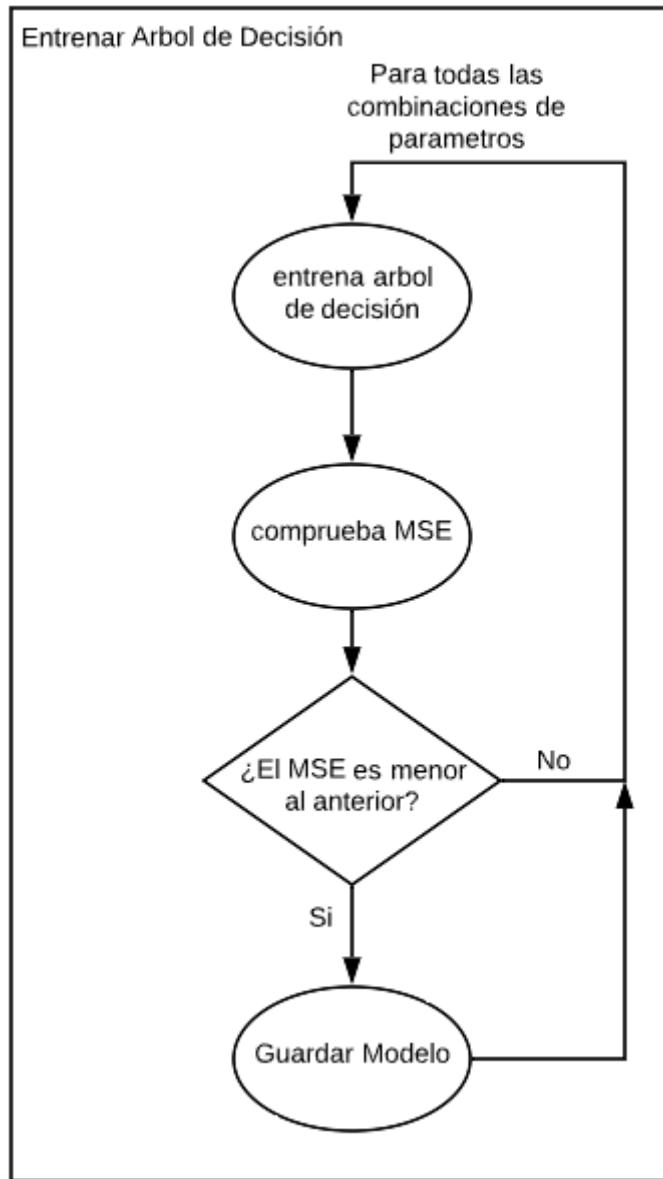


Imagen 8.4.4.1.2.2.4: diagrama entrenar Árbol de decisión

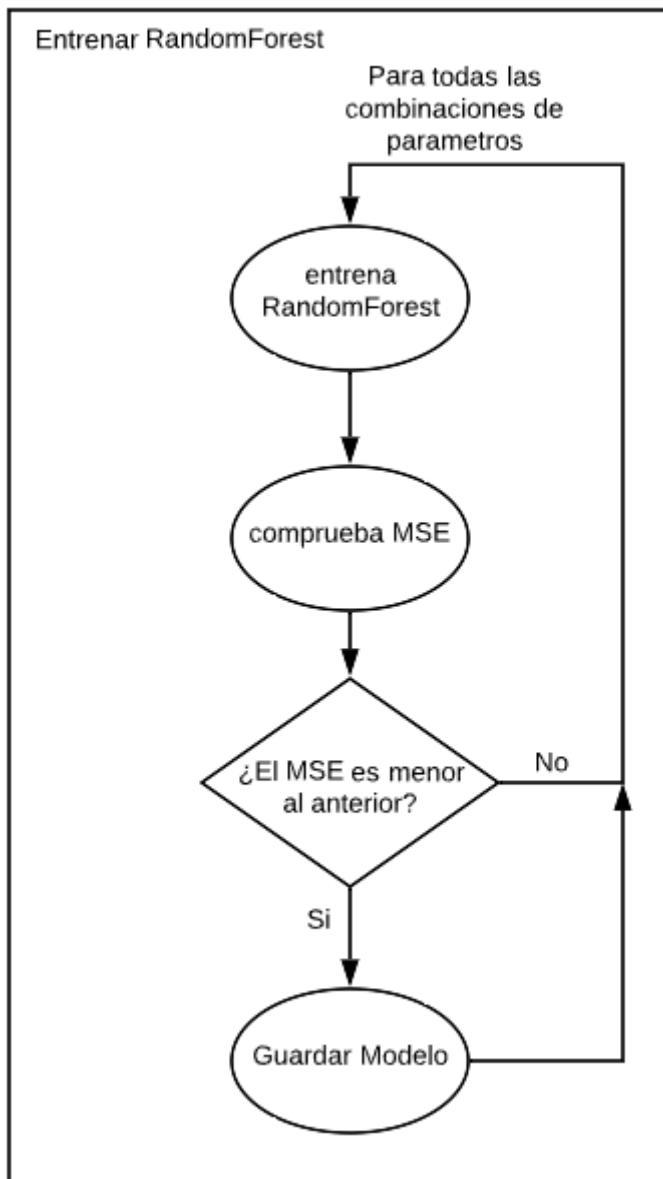


Imagen 8.4.4.1.2.2.5: diagrama entrenar RandomForest

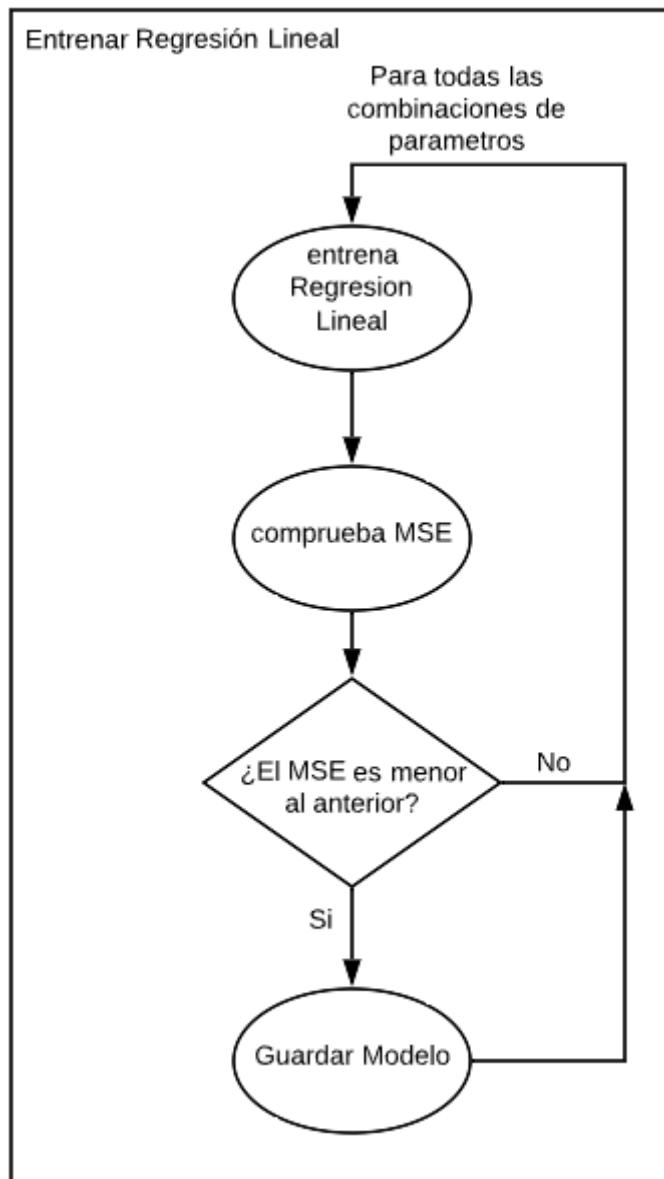


Imagen 8.4.4.1.2.2.6: diagrama entrenar Regresión Lineal

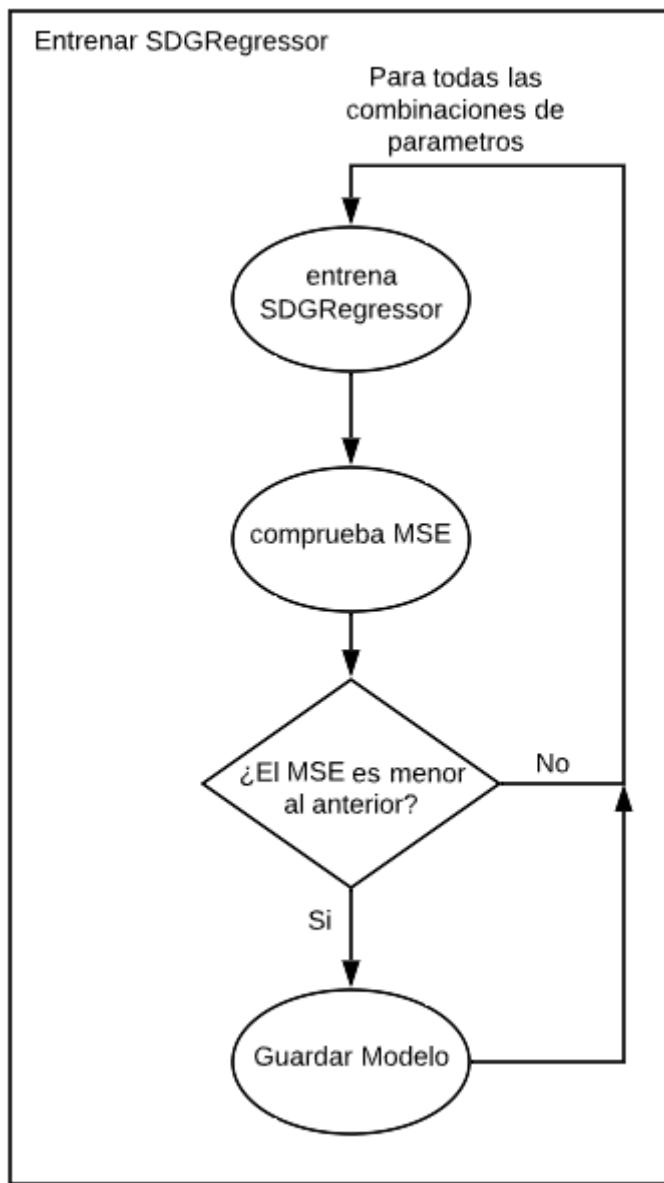


Imagen 8.4.4.1.2.2.7: diagrama entrenar SDGRegressor

También se va a crear un diagrama de flujo donde se pueda comprender de forma más sencilla el caso de uso de lectura y preprocesamiento de los datos. El diagrama es el siguiente:

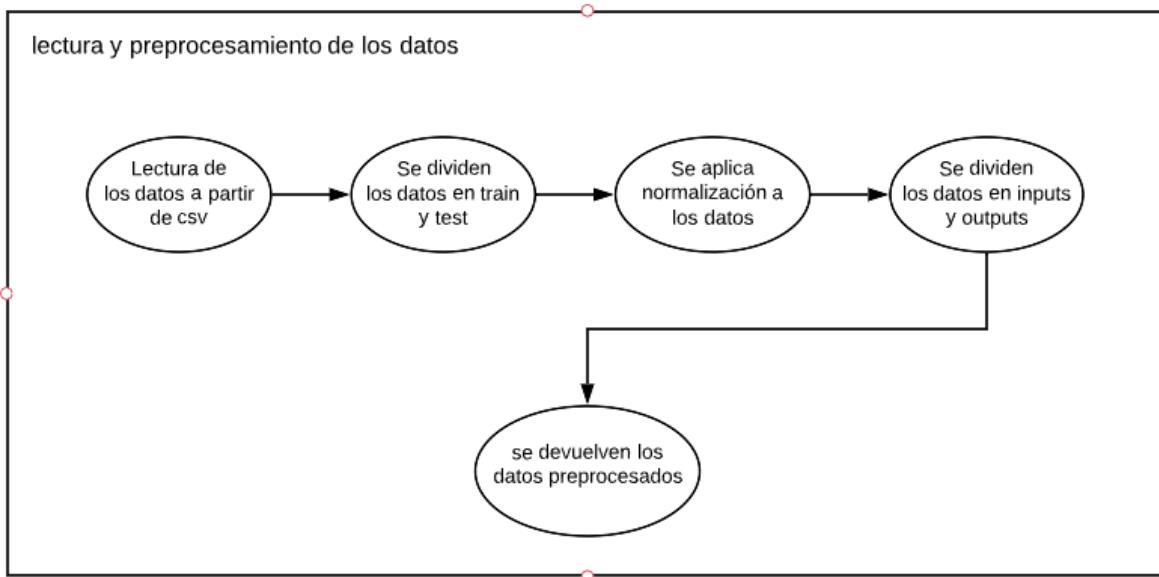


Imagen 8.4.4.1.2.2.8: diagrama de lectura y preprocesamiento de los datos

Dividir los datos en train y test, junto aplicar la normalización son necesarios para obtener el **MSE** correctamente como se vio en puntos anteriores. El hecho de dividir los datos en inputs y outputs se realiza para el correcto funcionamiento de las funciones de scikit a la hora de usar los métodos donde se aplican los algoritmos de los diferentes modelos. Cabe destacar, que antes de dividir los patrones es necesario transformar los atributos categóricos con **OneHotEncoder**, como se vio en el apartado de preprocesamiento. En resumen, este preprocesamiento es similar al de ese apartado.

#### 8.4.4.1.2.3 Predecir a partir de modelo

En este caso de uso se predice la iluminación superior a partir de un patrón recibido usando el modelo ya entrenado. Antes de comentar como funciona se va a explicar las entradas y salidas. El nombre de la petición es **predecir** y los parámetros de entrada que tiene son todos aquellos atributos que hay en un patrón siendo **ColorSuelo** de tipo str, **AlturaLuminaria** de tipo str, **FlujoLuminicoTotal** de tipo str, **TCC** de tipo str, **IluminanciaAbajo** de tipo str, **Espectro** de tipo str, **ReflectanciaSuelo** de tipo str y **IluminanciaSuperior** de tipo str, siendo todos ellos parámetros obligatorios. Un ejemplo de llamada para su mayor entendimiento, sería el siguiente si se realiza desde navegador (lo normal no es llamarlo directamente sino por la aplicación del cliente o el generador de gráficas):

<http://localhost:8000/predecir/?ColorSuelo=rojo&AlturaLuminaria=160&FlujoLuminicoTotal=3000&TCC=4000&IluminanciaAbajo=1600&Espectro=432&ReflectanciaSuelo=46&IluminanciaSuperior=0>

Cada uno de los atributos anteriormente descritos hace referencia al valor de cada atributo necesario para crear un patrón y poder predecir la iluminancia superior y por consiguiente su contaminación lumínica. Este get de la api si devuelve algo, en concreto un diccionario donde podemos encontrar la iluminancia superior para que pueda tener formato json y se pueda utilizar en el cliente.

El funcionamiento de dicho caso consiste en cargar el dataset del csv creado de la base de datos en el entrenamiento y cargar el modelo que dio mejores resultados. Se crea el patrón a partir de los datos pasados por la api y se aplica la predicción. Finalmente se devuelve la iluminancia superior.

Con este caso de uso queda explicado el servicio **modelo\_prediccion\_contaminacion\_luminica** completo, y todos los casos relacionados con el docker-compose. Ahora se explicarán los casos de usos de los programas auxiliares que no se incluyen en el docker.

Para mayor entendimiento, se ha creado el siguiente diagrama de flujo:

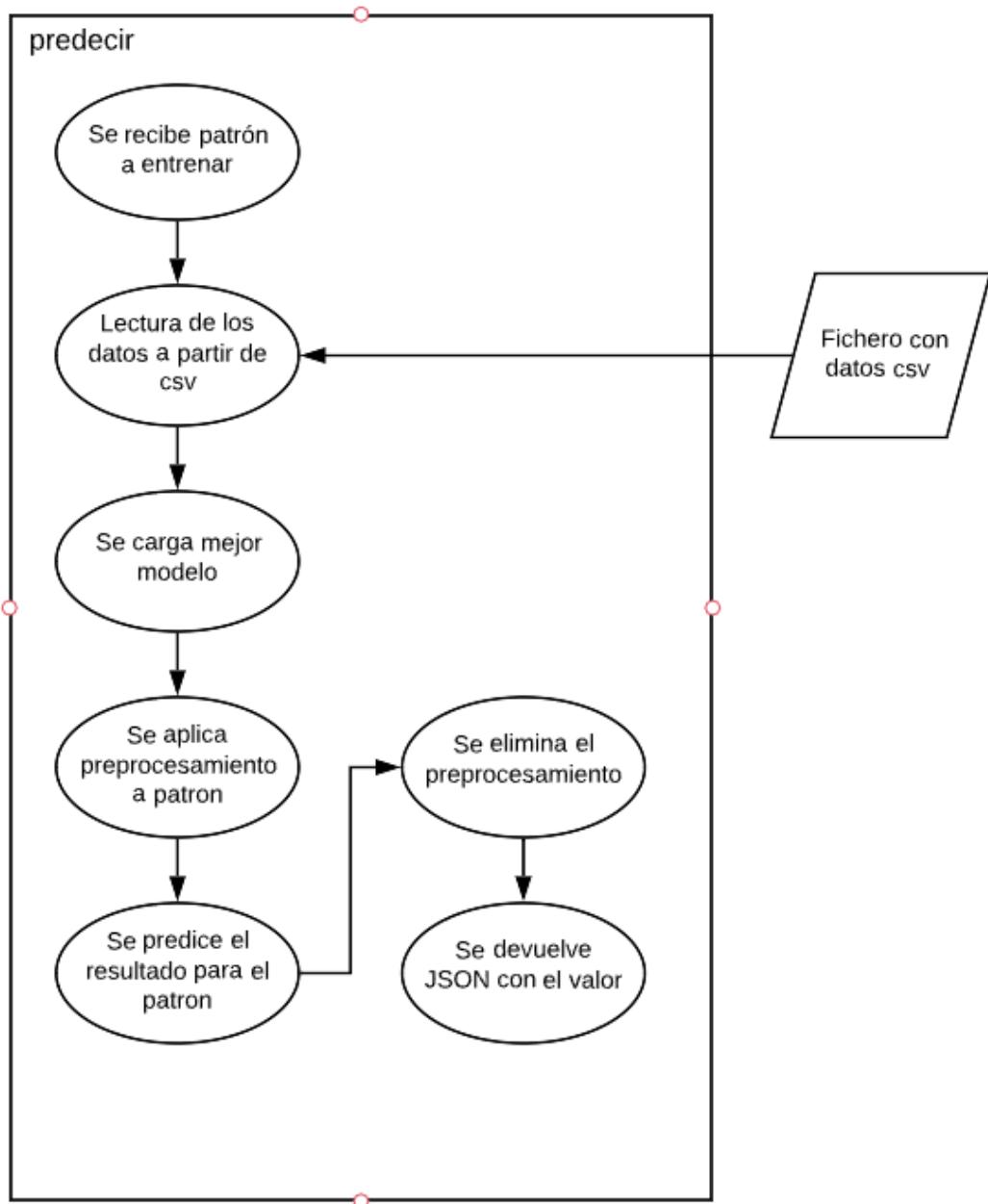


Imagen 8.4.4.1.2.3.1: diagrama de predicción

Este diagrama puede ser un poco más complejo ya que requiere de más casos de uso. El caso quizás menos entendible es el de cargar los datos del csv. Este caso es necesario ya que internamente, para poder aplicar el preprocesamiento es necesario juntar todos los patrones con el patrón nuevo y aplicarle así el preprocesamiento. Una vez realizado se separa del resto de patrones y se continúa con el resto de casos de uso.

#### 8.4.4.2. Programas auxiliares

En este apartado se explicarán aquellos programas creados que apoyan al sistema y que no se encuentran integrados en el mismo. Estos son **preprocesamiento**, **el ejemplo de cliente y generador de gráficas**.

##### 8.4.4.2.1. Preprocesamiento

Este caso de uso pretende englobar el preprocesamiento realizado para los datos obtenidos en el csv. Una vez los datos se encuentran en la bd se realizó este programa que permite obtener **histogramas** y el **mapa de correlaciones** de los datos obtenidos en el csv extrayéndolos de la bd. Para utilizarlo, únicamente hay que ejecutar el programa python y nos creará y guardará los histogramas y el mapa de correlaciones.

Para mayor entendimiento del funcionamiento del programa, se ha creado el siguiente diagrama de flujo:

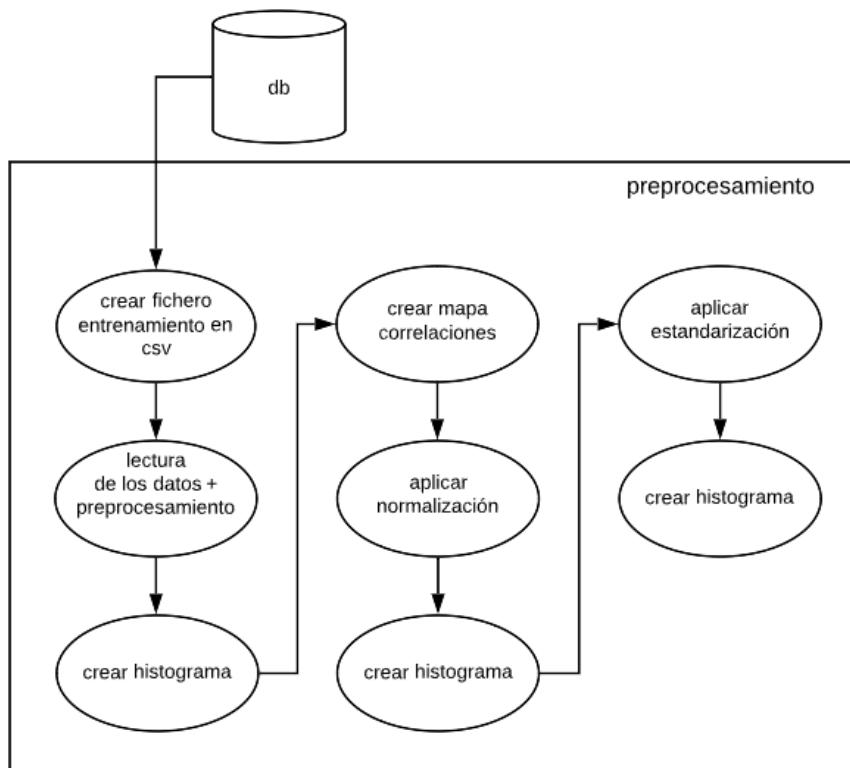


Imagen 8.4.4.2.1.1: diagrama de preprocesamiento

#### 8.4.4.2.2. Ejemplo Cliente

Este caso de uso incluye un programa sencillo que simula a un posible programa que permita al cliente realizar peticiones “get” de predicción de forma amigable, ofreciendo un resultado para los datos dados.

Para mayor entendimiento del funcionamiento del programa, se ha creado el siguiente diagrama de flujo:

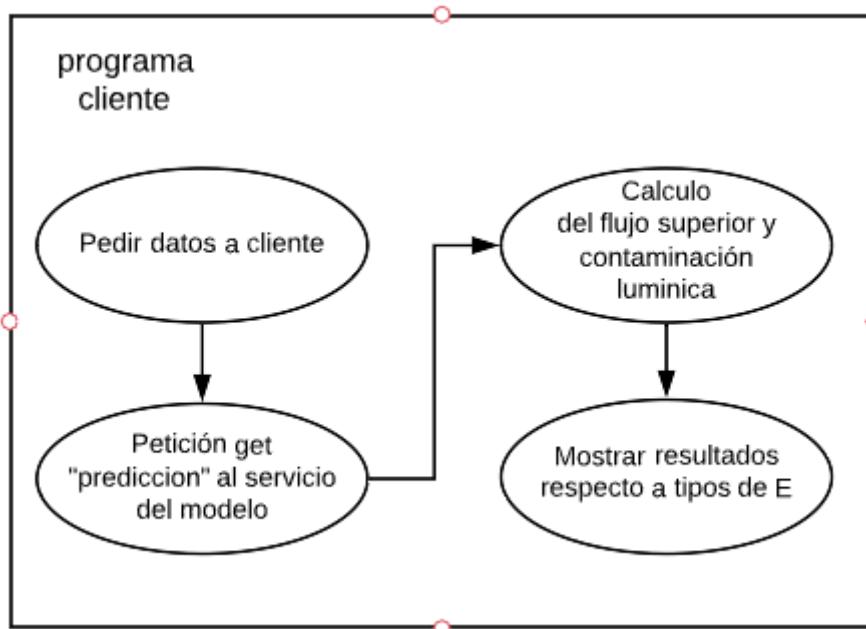


Imagen 8.4.4.2.2.1: diagrama del programa cliente

El caso de uso “Cálculo del flujo superior y contaminación lumínica”, se aplican las fórmulas vistas en el apartado 8.2.10. Flujo Lumínico Superior a Luminaria y FHSi

El caso de uso “Mostrar resultados respecto a tipos de E”, se refiere que respecto al porcentaje de contaminación lumínica recibido, indique, para la combinación de características recibida, en qué zonas sería posible su uso.

En puntos anteriores (en concreto el punto 8.1. Elección de características) se explica los distintos tipos de E.

#### 8.4.4.2.3. Generador Gráficas

Este caso de uso incluye un programa que permite crear una gráfica para comprobar el funcionamiento del mejor modelo frente a distintos patrones y comprender mejor los distintos resultados.

Para su correcto funcionamiento, se deben extraer una cierta cantidad de patrones del fichero csv para comprobar los resultados que nos ofrece el modelo. Si se usaran esos patrones, el modelo estaría haciendo trampa ya que los conoce porque ha entrenado con ellos. Por ello, la creación de la flag que evita crear el fichero a partir de la base de datos.

Para mayor entendimiento del funcionamiento del programa, se ha creado el siguiente diagrama de flujo:

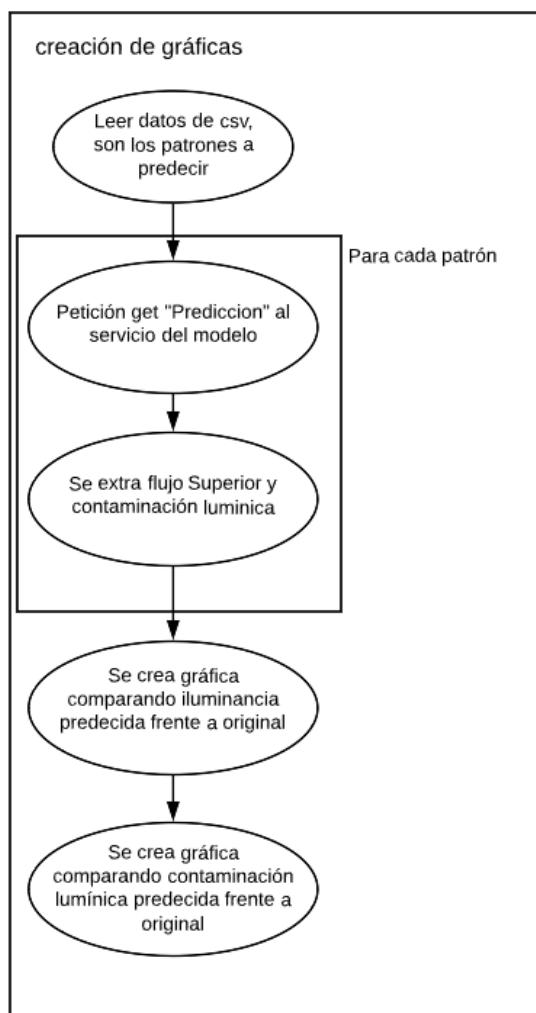
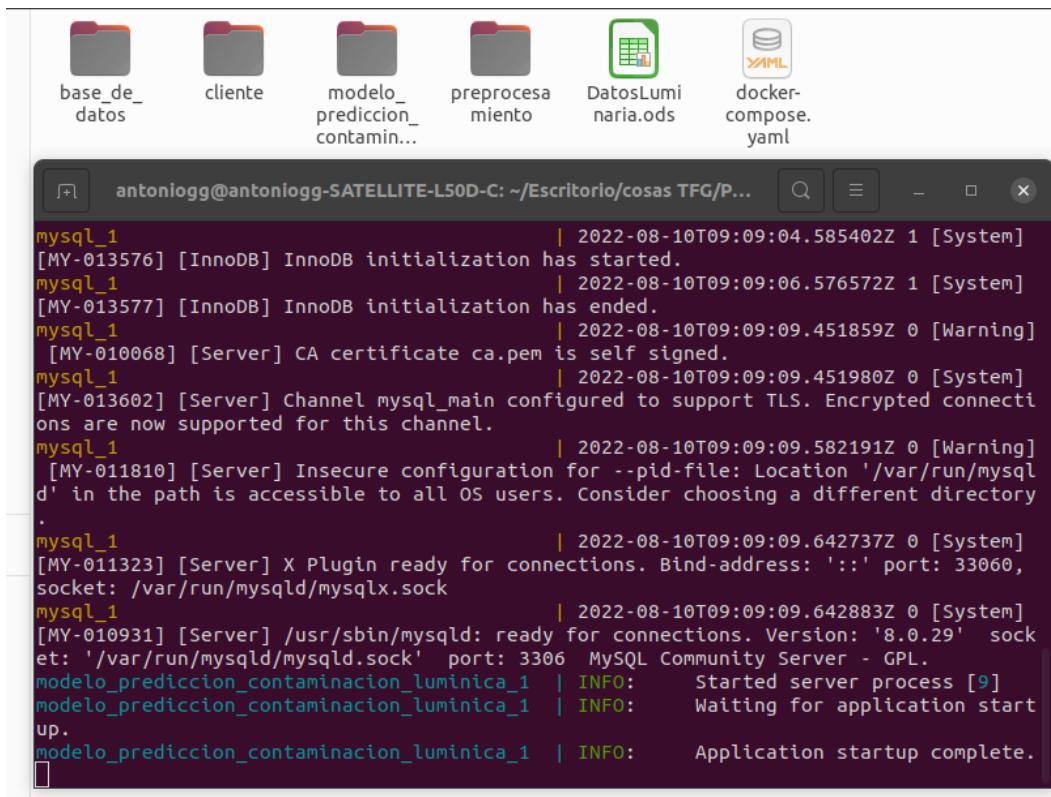


Imagen 8.4.4.2.3.1: diagrama del generador de gráficas

#### 8.4.4.3. Simulación de uso y resultados obtenidos

Una vez visto los modelos, como comparamos los distintos modelos con el uso del **MSE** y la estructura de todo el sistema, más los programas de apoyo; Se pretende realizar una simulación donde se comprobará todo y se analizarán los distintos resultados para obtener una conclusión.

Primero, se accede al directorio donde se encuentra el docker-compose.yaml para lanzar los servicios del docker-compose con el comando **docker-compose up** del cmd.



```
mysql_1           | 2022-08-10T09:09:04.585402Z 1 [System]
[MY-013576] [InnoDB] InnoDB initialization has started.
mysql_1           | 2022-08-10T09:09:06.576572Z 1 [System]
[MY-013577] [InnoDB] InnoDB initialization has ended.
mysql_1           | 2022-08-10T09:09:09.451859Z 0 [Warning]
[MY-010068] [Server] CA certificate ca.pem is self signed.
mysql_1           | 2022-08-10T09:09:09.451980Z 0 [System]
[MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
mysql_1           | 2022-08-10T09:09:09.582191Z 0 [Warning]
[MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysql/d' in the path is accessible to all OS users. Consider choosing a different directory
.
mysql_1           | 2022-08-10T09:09:09.642737Z 0 [System]
[MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
mysql_1           | 2022-08-10T09:09:09.642883Z 0 [System]
[MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.29' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.
modelo_prediccion_contaminacion_luminica_1 | INFO:     Started server process [9]
modelo_prediccion_contaminacion_luminica_1 | INFO:     Waiting for application start up.
modelo_prediccion_contaminacion_luminica_1 | INFO:     Application startup complete.
```

Imagen 8.4.4.3.1: ejemplo consola de comandos arranque

Una vez los servicios se encuentren funcionales, ya podemos realizar llamadas a la **API** a través del puerto 8000. Se realizaría un inserción como se explicó anteriormente para que los datos del csv se almacenen en la bd.

Una vez los datos están en la bd, podríamos realizar el preprocesamiento. No se explican los resultados obtenidos porque se explicaron en el apartado de preprocesamiento.

Si se desea entrenar el modelo para poder predecir patrones se llamaría a la api como anteriormente se comentó, generando los siguientes resultados para cada modelo:

### **Resultados SVR:**

Como se puede observar en la siguiente imagen, encontramos en la primera columna el **MSE** y en la segunda el **MAE** para cada combinación entrenada. En la última línea se especifica qué combinación ha dado un **MSE** más bajo guardando así ese modelo con el scaler que transformó los datos (el escalado para en la predicción poder desescalar los datos)

```
26 MSE & MAE Final con C=100.000000 y G=0.100000: 0.002618 0.042008
27 MSE & MAE Final con C=100.000000 y G=1.000000: 0.007757 0.080983
28 MSE & MAE Final con C=100.000000 y G=10.000000: 0.011394 0.097489
29 MSE & MAE Final con C=100.000000 y G=100.000000: 0.012363 0.100461
30 MSE & MAE Final con C=100.000000 y G=1000.000000: 0.012411 0.100855
31 MSE & MAE Final con C=1000.000000 y G=0.100000: 0.002769 0.042618
32 MSE & MAE Final con C=1000.000000 y G=1.000000: 0.002618 0.042008
33 MSE & MAE Final con C=1000.000000 y G=10.000000: 0.007757 0.080983
34 MSE & MAE Final con C=1000.000000 y G=100.000000: 0.011394 0.097489
35 MSE & MAE Final con C=1000.000000 y G=1000.000000: 0.012363 0.100461
36 MSE & MAE Final con C=1000.000000 y G=10000.000000: 0.012411 0.100855
37 Los mejores parametros son C:10.0 v Gamma:0.1 con un MSE de 0.0025800255120222394 v un MAE de 0.041812274462622275
```

Imagen 8.4.4.3.2: resultados SVR

La mejor combinación el C=10 con Gamma= 0.1 con un MSE de 0.0025

### **Resultados Árbol de decisión:**

Como se puede observar en la siguiente imagen, encontramos en la primera columna el **MSE** y en la segunda el **MAE** para cada combinación de parámetros del modelo. En la última línea se especifica la combinación que ha dado un **MSE** más bajo, guardando así ese modelo con el scaler que transformó los datos.

```
121 MSE & MAE Final con splitter=random, min_samples_split=0.800000 y min_samples_leaf=0.100000: 0.020414 0.109968
122 MSE & MAE Final con splitter=random, min_samples_split=0.800000 y min_samples_leaf=0.400000: 0.020414 0.109968
123 MSE & MAE Final con splitter=random, min_samples_split=0.800000 y min_samples_leaf=0.300000: 0.020414 0.109968
124 MSE & MAE Final con splitter=random, min_samples_split=0.800000 y min_samples_leaf=0.200000: 0.020414 0.109968
125 MSE & MAE Final con splitter=random, min_samples_split=0.800000 y min_samples_leaf=0.500000: 0.023063 0.122572
126 MSE & MAE Final con splitter=random, min_samples_split=0.800000 y min_samples_leaf=0.600000: 0.023063 0.122572
127 MSE & MAE Final con splitter=random, min_samples_split=0.800000 y min_samples_leaf=0.700000: 0.023063 0.122572
128 MSE & MAE Final con splitter=random, min_samples_split=0.800000 y min_samples_leaf=0.800000: 0.023063 0.122572
129 Los mejores parametros para el arbol de decision es splitter->best, min_samples_split->0.3, min_samples_leaf->0.1 con un MSE de: 0.007585239887971149 v un MAE de 0.0556335697438181545
```

Imagen 8.4.4.3.3: resultados Árbol de decisión

La mejor combinación para este modelo sería splitter=best, min\_samples\_split=0.3 y min\_samples\_leaf=0.1 dando un MSE de 0.0075.

### **Resultados Random Forest:**

En la siguiente imagen, encontramos en la primera columna el **MSE** y en la segunda el **MAE** para cada combinación de parámetros. En la última línea se especifica la combinación que ha dado un **MSE** más bajo, guardando así ese modelo con el scaler que transformó los datos.

```
84 MSE & MAE Final con n_estimators=300, v_max_features=0.010000, min_samples_split=100.000000, min_samples_leaf=50.000000= 0.020737
  0.116124
85 MSE & MAE Final con n_estimators=300, v_max_features=0.010000, min_samples_split=2.000000, min_samples_leaf=100.000000= 0.023114
  0.122864
86 MSE & MAE Final con n_estimators=300, v_max_features=0.010000, min_samples_split=2.000000, min_samples_leaf=2.000000= 0.019049
  0.111411
87 MSE & MAE Final con n_estimators=300, v_max_features=0.010000, min_samples_split=2.000000, min_samples_leaf=50.000000= 0.020257
  0.114254
88 MSE & MAE Final con n_estimators=300, v_max_features=0.010000, min_samples_split=50.000000, min_samples_leaf=100.000000= 0.023038
  0.122430
89 MSE & MAE Final con n_estimators=300, v_max_features=0.010000, min_samples_split=50.000000, min_samples_leaf=2.000000= 0.018273
  0.108821
90 MSE & MAE Final con n_estimators=300, v_max_features=0.010000, min_samples_split=50.000000, min_samples_leaf=50.000000= 0.020558
  0.115179
91 Los mejores parametros para el random forest es n_estimators->300max_features->0.3, min_samples_split->2, min_samples_leaf->2 con un MSE
de:0.009173122584899562 y un MAE de 0.074343008715534958
```

Imagen 8.4.4.3.4: resultados RandomForest

En este caso la mejor combinación es la siguiente donde n\_estimators=300, max\_features=0.3, min\_samples\_split=2 y min\_samples\_leaf=2 dando como MSE 0.0091

### **Resultados Regresión Lineal:**

En la siguiente imagen, se diferencia de las anteriores, ya que en este caso no hay distintas combinaciones de parámetros para este modelo, únicamente se entrenaba una vez viendo el **MSE** que nos ofrecía.

```
1 MSE & MAE Final con para la regresion lineal: 0.007025 0.067623
```

Imagen 8.4.4.3.5: resultados Regresión Lineal

El MSE dado por este modelo es de 0.007

### Resultados SDGRegressor:

En la siguiente imagen, similar a los modelos anteriores, encontramos en la primera columna el **MSE** y en la segunda el **MAE** para cada combinación de parámetros. En la última línea se especifica la combinación que ha dado un **MSE** más bajo, guardando así ese modelo con el scaler que transformó los datos.

Cabe destacar que es el modelo que realizó más entrenamiento con distintos tipos de características.

629 MSE & MAE Final con early_stopping=True, validation=0.200000, shuffle=True, alpha=0.001500=	0.006064 0.065027
630	
631 MSE & MAE Final con early_stopping=True, validation=0.200000, shuffle=True, alpha=0.001600=	0.006900 0.067789
632	
633 MSE & MAE Final con early_stopping=True, validation=0.200000, shuffle=True, alpha=0.001700=	0.006947 0.070029
634	
635 MSE & MAE Final con early_stopping=True, validation=0.200000, shuffle=True, alpha=0.001800=	0.006618 0.069144
636	
637 MSE & MAE Final con early_stopping=True, validation=0.200000, shuffle=True, alpha=0.001900=	0.007042 0.070500
638	
639 MSE & MAE Final con early_stopping=True, validation=0.200000, shuffle=True, alpha=0.002000=	0.006157 0.067152
640	
641 Los mejores parametros para el SGDRegressor es early_stopping->True, validation->0.4, shuffle->True, alpha->0.0006 con un MSE de: 0.005357486809333842 v un MAE de 0.05835813839758818	

Imagen 8.4.4.3.6: resultados SDGRegressor

La mejor combinación de parámetros para el SDGRegressor es `early_stopping=True, validation=0.4, shuffle=True` y `alpha=0.0006` con un MSE de 0.0053

Una vez vistos todos los **MSE** podemos comparar entre sí, con la siguiente tabla se puede ver mejor la comparativa:

SVR	Árbol de decisión	Random Forest	Regresión Lineal	SDGRegressor
0.0025	0.0075	0.0091	0.007	0.0053

Como podemos ver en la tabla el mejor modelo es **SVR** y el peor es **Random Forest**. Cabe destacar que siempre hay que tener en cuenta que dependiendo de los datos unos modelos pueden ser mejores que otros de tal forma que si se tienen mayor cantidad de datos, alguno de los otros modelos puede superar a **SVR**. También recalcar que se lanzó varias veces el entrenamiento para comprobar si salían **MSE** similares, suelen dar los mismos valores pero al tener ciertos factores aleatorios, como la partición de los datos de entrenamiento en un 90% entrenamiento y 10% test, puede tanto aumentar como disminuir el MSE de los distintos modelos.

Recalcamos que estos modelos se realizaron con la normalización, ya que daban resultados muy parecidos y se escogió finalmente normalización de los datos.

Si razonamos el porqué del **MSE** de cada uno se pueden entender ciertas cosas como por ejemplo que el **MSE** de **SDGRegressor** sea más bajo que el de Regresión Lineal, esto es lógico ya que al realizar más pruebas con distintas combinaciones tiende a bajar más el **MSE** que la Regresión Lineal, ya que esta solo realiza un entrenamiento.

Sorprendentemente, **SVR** nos da mejores resultados, aun siendo un modelo más orientado a clasificación.

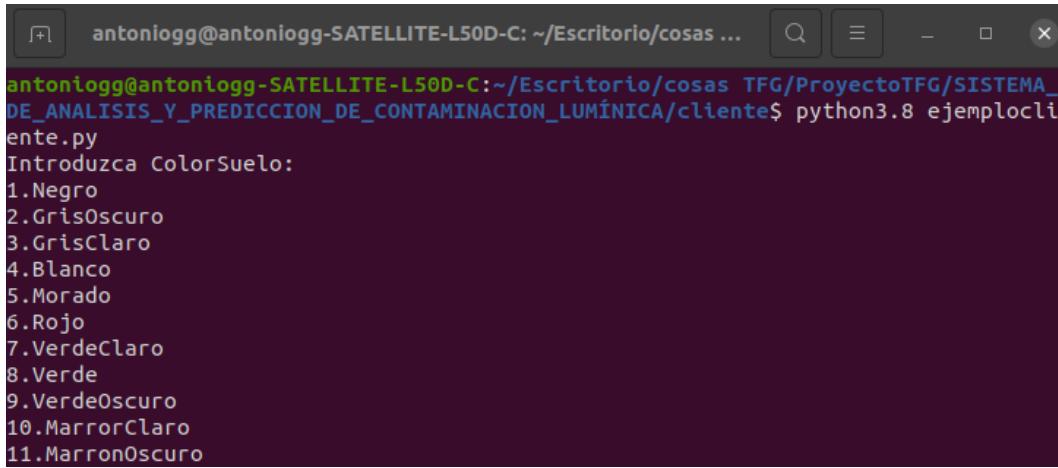
Realizando pruebas con otras bases de datos, da la sensación de que mientras más pequeña sea la cantidad de patrones, mejor es el entrenamiento que realiza, dando mejores resultados (aun así no perfila también como otros modelos con más datos).

Cabe destacar, que curiosamente, el random forest genera peor **MSE** que el árbol de decisión, incluso siendo un ensemble de varios de estos. Esto demuestra, que por usar más modelos no implica que vaya a generar mejores resultados, ya que puede generar más ruido, eso no quita que con mayor cantidad de patrones de mejores resultados en un futuro.

Finalmente, tras ver que el **SVR** da mejor **MSE**, se almacena como el mejor modelo, siendo este el utilizado en la predicción. La predicción como tal, se limita al recibir un patrón dar la predicción de la iluminancia superior, por ello, se pasa a explicar el sencillo programa que utilizaría un cliente para realizar peticiones.

El siguiente programa, es un programa de **python** que se lanzaría en consola. En ese programa, se van pidiendo los distintos datos del patrón al usuario y finalmente se le da el resultado obtenido por el predictor. A parte se le incluye el porcentaje de **contaminación lumínica** para la combinación de parámetros elegida y en qué tipo de **E** se incluye.

En la siguiente imagen se pueden ver algunos ejemplos de cómo el programa pide los datos al cliente.



```
antonioogg@antoniogg-SATELLITE-L50D-C: ~/Escritorio/cosas TFG/ProyectoTFG/SISTEMA_DE_ANALISIS_Y_PREDICCIÓN_DE_CONTAMINACION_LUMÍNICA/cliente$ python3.8 ejemplocliente.py
Introduzca ColorSuelo:
1.Negro
2.GrisOscuro
3.GrisClaro
4.Blanco
5.Morado
6.Rojo
7.VerdeClaro
8.Verde
9.VerdeOscuro
10.MarronClaro
11.MarronOscuro
```

Imagen 8.4.4.3.7: petición al cliente, se pide color del suelo



```
antonioogg@antoniogg-SATELLITE-L50D-C: ~/Escritorio/cosas TFG/ProyectoTFG/SISTEMA_DE_ANALISIS_Y_PREDICCIÓN_DE_CONTAMINACION_LUMÍNICA/cliente$ python3.8 ejemplocliente.py
Introduzca FlujoLuminicoTotal: 3500
```

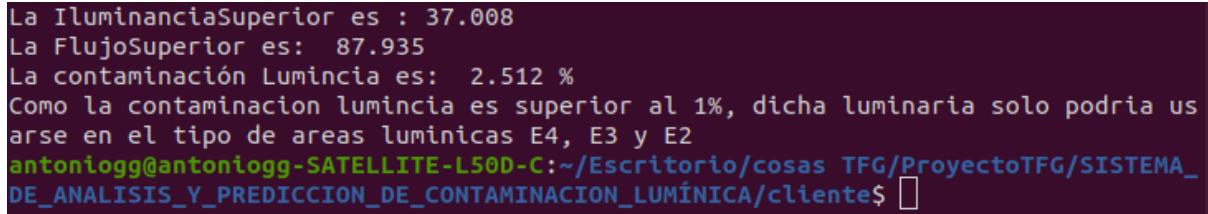
Imagen 8.4.4.3.8: petición al cliente, se pide flujo lumínico total

Por lo tanto, si introducimos los siguientes parámetros:

Color del suelo->negro  
Altura de la luminaria->80.0  
Flujo luminoso total->3500  
Temperatura correlada del color->4144  
Iluminancia abajo->1473  
Espectro->442  
Reflectancia->4

El patrón original con esta combinación de parámetros es 35.56. Por tanto, este valor debería de ser el resultado que debería obtenerse (o un valor cercano).

El resultado que obtendríamos sería el siguiente:



```
La IluminanciaSuperior es : 37.008
La FlujoSuperior es: 87.935
La contaminación Lumincia es: 2.512 %
Como la contaminacion lumincia es superior al 1%, dicha luminaria solo podria usarse en el tipo de areas luminicas E4, E3 y E2
antonioogg@antoniogg-SATELLITE-L50D-C:~/Escritorio/cosas TFG/ProyectoTFG/SISTEMA_DE_ANALISIS_Y_PREDICCIÓN_DE_CONTAMINACION_LUMÍNICA/cliente$
```

Imagen 8.4.4.3.9: salida respecto al patrón introducido por el cliente

Tanto el Flujo superior como la contaminación lumínica se obtienen con las fórmulas explicadas anteriormente.

Si nos fijamos en la imagen, la iluminancia superior es 37.008, esto implica que el flujo superior es 87.935, de tal forma que la contaminación lumínica es de un 2.5% permitiendo áreas E4, E3 y E2 ya que el índice obtenido es mayor que 1% pero menor a 5%.

El patrón introducido se ha hecho con trampa, ya que es un patrón que ya se encontraba dentro de los patrones de entrenamiento. La iluminancia superior de este patrón era de 35.56 que frente al 37 que hemos obtenido está bastante cercano. Aun así para evitar esta trampa y comprobar cómo funcionaba frente a patrones desconocidos se pensó en lo siguiente.

Crear un programa que generará un gráfica con distintos patrones que no se encontraran en el entrenamiento, por ello, del csv que genera el propio programa se le extrajeron 11 patrones, uno de cada color y se añadieron a un csv temporal. De esta forma llamamos a la api de entrenar pero con la flag de crear un nuevo csv en false para que entrene sin esos patrones de la base de datos. Una vez realizado el entrenamiento se realiza la predicción pasándole los patrones de el csv temporal y comparamos en una gráfica los resultados obtenidos con los reales, dando las siguientes gráficas.

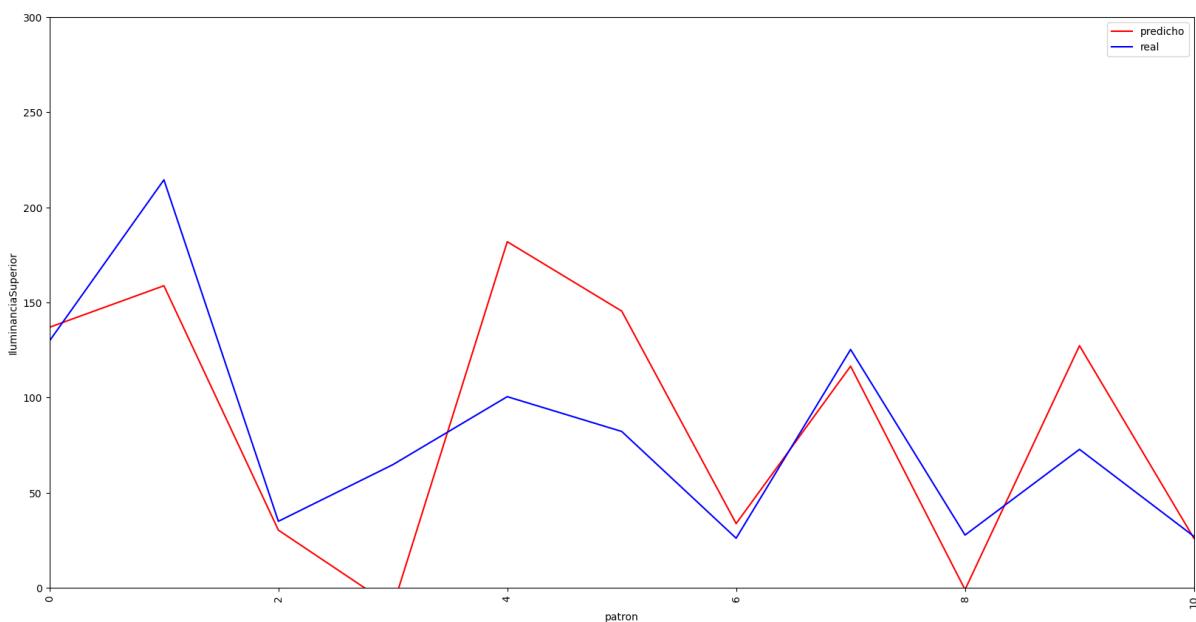


Imagen 8.4.4.3.10: gráfica comparativa entre la iluminancia superior predicha y la real

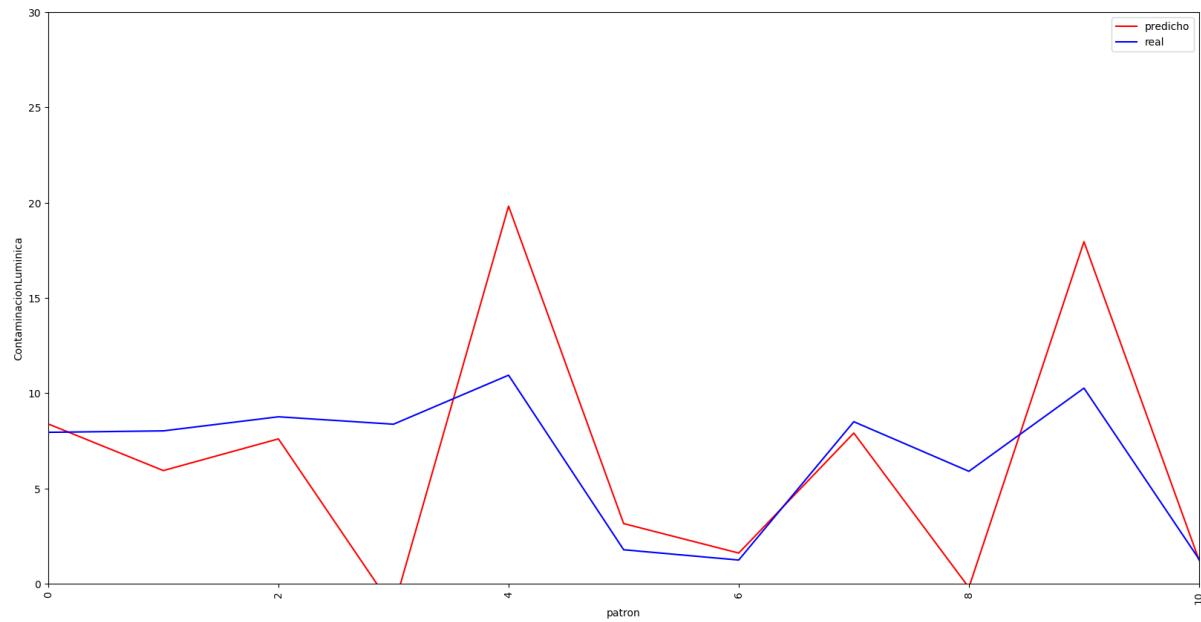


Imagen 8.4.4.3.11: gráfica comparativa entre la contaminación lumínica superior predicha y la real

La primera gráfica pertenece a la iluminancia superior obtenida frente a la iluminancia superior real, la segunda gráfica es similar pero con la contaminación lumínica producida (aplicando la fórmula).

Como podemos observar hay muchos valores que son similares, esto indica que el modelo predice de forma correcta. Aun así existen valores que difieren bastante y de hecho, dan valores que carecen de sentido siendo 0 o un valor negativo. Esto tiene su explicación y se debe a la cantidad tan reducida de patrones existentes en el entrenamiento.

Si retomamos los tipos de patrones que tenemos, existen dos por combinación, si uno de los dos se extrae del entrenamiento para que se use como test aquí en la gráfica, implica que solo tiene uno para entrenar y aparte, puede dar la casualidad de que ese patrón no se use en el 90% de entrenamiento y que se encuentre en el 10% de test, por lo tanto no tenemos ningun patron de ese estilo y el modelo no va a saber cómo predecir esos tipos de patrones.

Aun así para tener 333 patrones de entrenamiento, que si quitamos 11 para la gráfica, son 222, se usan solo el 90% para entrenar y aun así. De 11 patrones desconocidos ha sido capaz de predecir casi correctamente 9. Por lo tanto consideramos que el modelo dentro de un entorno controlado y de laboratorio, da buenos resultados.



## 9. Pruebas:

En este apartado se realizan pruebas para comprobar el correcto funcionamiento del sistema. Primeramente se revisarán los requerimientos funcionales y no funcionales del sistema para comprobar si necesitarán pruebas. Como recordatorio los **requisitos funcionales y no funcionales** son los siguientes:

**RFUN - 1:** La base de datos debe ser creada con mysql.

**RFUN - 2:** La base de datos debe tener creada una tabla donde se encuentren todos los atributos con los que se pretende trabajar (a la hora de usar estos atributos no se tiene que ver obligado a usarlos todos).

**RFUN - 3:** Los servicios deben estar encapsulados en docker para poder usarlo dentro de un docker-compose.

**RFUN - 4:** El sistema a la hora de entrenar debe ser capaz de almacenar los mejores modelos para cada combinación y el mejor modelo en general. Cabe destacar que deberá almacenar el preprocesamiento realizado para los datos ya que es necesario para quitar ese preprocesamiento a posteriori.

**RFUN - 5:** El sistema a la hora de entrenar deberá almacenar en distintos ficheros de texto los resultados obtenidos para cada modelo para cada tipo de combinación.

**RFUN - 6:** El servicio ofrecido por el docker donde se encuentran los modelos, debe responder peticiones “get”, siendo inserción, entrenamiento y predicción.

**RFUN - 7:** La devolución del resultado de la predicción debe encontrarse en formato JSON para mayor facilidad de uso.

**RFUN - 8:** El sistema una vez se inicia a través del docker compose, debe ser resistente a fallos en sus peticiones, evitando que caiga todo el sistema.

**RFUN - 9:** El sistema dará información de las peticiones que recibe a través de la consola con la que fue iniciado, además cada vez que termine de realizar una petición devolverá un mensaje de éxito o error (se excluye el caso de la predicción que devuelve un JSON).

**RNFUN - 1:** El sistema global está formado por dos servicios (la base de datos y el servicio que contiene los modelos), estos deben ser claramente diferenciables, siendo de esta manera un sistema modular y fácilmente mantenable.

**RNFUN - 2:** El sistema al ser modular (al realizarse con docker), debe permitir mejoras en el futuro y capacidad de añadir nuevos servicios. Esto permite que el sistema sea un sistema escalable.

**RNFUN - 3:** Al introducir ficheros que no son csv el sistema debe detectarlos y enviar un mensaje que avise del problema.

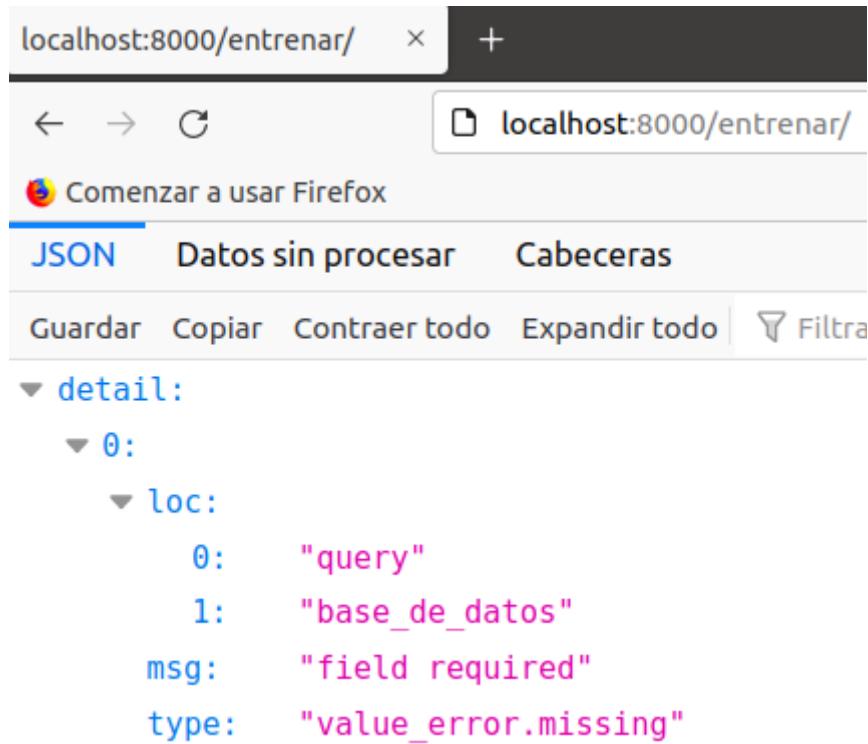
**RNFUN - 4:** Cuando una petición se ha realizado de forma errónea, debido a los parámetros introducidos, el sistema debe devolver un mensaje que avise de dicho problema.

**RNFUN - 5:** Cada vez que se vuelve a lanzar el entrenamiento, deben de eliminarse todos los tipos de modelos guardados con anterioridad, el caso de mejor modelo, sus respectivos preprocesamientos y los ficheros de texto plano con la información de cada prueba y ser sustituidos por los nuevos resultados.

Como podemos ver, la mayoría de requerimientos no necesitan pruebas, ya que muchas vienen dada por el diseño e incluso durante el propio desarrollo del apartado “**8. Metodología**”, se van comprobando que se cumplen estos requisitos.

Aun así nos podemos encontrar ciertos requerimientos que sí se pueden comprobar como son los siguientes:

Los requerimientos **RFUN - 8** y **RNFUN - 3**, se pueden comprobar de forma sencilla al introducir una petición “get” errónea para ver cómo se comporta el sistema y si deja completamente de funcionar el mismo. En la imagen siguiente se pueden apreciar los resultados



```
localhost:8000/entrenar/ × +  
← → C localhost:8000/entrenar/  
Comenzar a usar Firefox  
JSON Datos sin procesar Cabeceras  
Guardar Copiar Contraer todo Expandir todo Filtra  
▼ detail:  
▼ 0:  
▼ loc:  
0: "query"  
1: "base_de_datos"  
msg: "field required"  
type: "value_error.missing"
```

Imagen 9.1.: salida de petición incompleta/errónea

Como se puede apreciar en la imagen anterior, el sistema avisa de que la petición no se valida y sigue permitiendo la llamada de distintas peticiones entrenar y predecir, por ejemplo.

Respecto al requerimiento **RNFUN - 3**, se comprobó si se cumplía este requerimiento al introducir un archivo txt. El resultado fué el siguiente:



```
localhost:8000/insertarPatrones × +  
localhost:8000/insertarPatrones/?base_de_datos=db&archivo_datos=prueba.txt  
← → C  
Comenzar a usar Firefox  
JSON Datos sin procesar Cabeceras  
Guardar Copiar Contraer todo Expandir todo Filtrar JSON  
0: "Error con el fichero de datos introducido"
```

Imagen 9.2.: salida producida tras utilizar un fichero txt

Como se puede apreciar en la imagen anterior, se cumple dicho requerimiento. Por tanto, podemos afirmar que **se cumplen** todos los requerimientos anteriores. Aun así, se comentaron también en otros puntos, como el de los **3.Objetivos**, que el sistema **debe ser relativamente rápido**. Para ello se realizaron pruebas comprobando los tiempos que nos da el sistema para cada petición “get”. También se comentó que el sistema debe ser capaz de **dar resultados aceptables**, ese apartado también se comentará a continuación.

Respecto a los tiempos, se pretende que sea un sistema que tenga unos tiempos aceptables. Es lógico que a la hora de entrenar se demore el sistema, pero cuando un cliente pida averiguar la contaminación lumínica de un patrón, este debe ser casi instantáneo. Los tiempos obtenidos para cada petición “get” son los siguientes:

```
modelo_prediccion_contaminacion_luminica_1 | El tiempo total transcurrido del proceso de insercion  
fue: 56.38665843009949
```

Imagen 9.3.: tiempo de inserción

```
modelo_prediccion_contaminacion_luminica_1 | El tiempo total transcurrido del proceso de  
entrenamiento fue: 102.79824829101562
```

Imagen 9.4.: tiempo de entrenamiento

```
El tiempo transcurrido del proceso fue: 0.0553898811340332
```

Imagen 9.5.: tiempo de predicción

Como se puede observar, a la hora de **insertar patrones** de un fichero de texto a la base de datos, el tiempo obtenido es de **56.38 segundos**.

Esto es algo aceptable y lógico ya que las transacciones a la base de datos suelen ser relativamente lentas. Cabe destacar que a mayor cantidad de datos que se pretenda introducir en la base de datos, mayor será el tiempo que tardará en insertar los patrones.

Respecto al **entrenamiento**, se realizó la prueba obteniendo el valor de **102.79 segundos**.

Este tiempo es más que aceptable, ya que el proceso de entrenamiento, es el proceso que conlleva más tiempo de todo el sistema y donde se están entrenando distintos modelos.

Cabe destacar que mientras más modelos se pretendan probar y mayor sea la cantidad de patrones, mayor será la cantidad de tiempo que tarda en dar los resultados. Por ello se buscó un equilibrio entre número de patrones y tiempo de respuesta del sistema.

Por último, se realizó la prueba al realizar una petición de predicción, obteniendo como resultado un tiempo de 0.055 segundos. Este tiempo es prácticamente instantáneo, logrando así los objetivos dichos anteriormente. Esto se debe, a que como el mejor modelo ya se encuentra entrenado y almacenado, únicamente se limita a cargarlo y predecir el patrón dado por el usuario.

Respecto al tema de la fiabilidad del sistema a la hora de dar los resultados predichos, como se vio en el apartado “**8.4.4.3. Simulación de uso y resultados obtenidos**”, se explican los resultados obtenidos y de hecho, se crea un programa auxiliar que permite demostrar, cuán fiables son los resultados generados por el modelo. Para ello, se crearon las gráficas **8.4.4.3.10** y **8.4.4.3.11**, donde se muestra en dichas gráficas una comparativa entre los resultados que se esperaban obtener frente a los valores dados por el sistema.

De dichas gráficas, se puede sacar en conclusión que la fiabilidad del sistema es aceptable, pero es lógico que con una mayor cantidad de patrones se podría obtener un mejor resultado. De hecho, se podrían realizar mayor cantidad de pruebas, ya que, si se realizan pruebas con un mayor número de patrones, la calidad del entrenamiento se disminuiría por la disminución de patrones de entrenamiento.



## 10. Conclusiones y futuras mejoras:

Como se ha podido apreciar durante la documentación de este tfg y más concretamente en el apartado anterior. Se podría decir que se han conseguido casi todos los objetivos marcados, de hecho, se van a revisar los objetivos de uno en uno para comprobar si se han cumplido.

En la siguiente tabla se muestra un resumen de los objetivos y, de si se han conseguido lograr, añadiendo unas observaciones de los mismos.

<b>Objetivo</b>	<b>Consecución</b>	<b>Observaciones</b>
OB-01	Sí	Se ha estudiado y aprendido conceptos sobre la caracterización de luminarias. Aprendiendo en el proceso sobre un campo desconocido para mí.
OB-02	Sí	Se ha conseguido montar una estructura y metodología necesaria para poder obtener los datos.
OB-03	Parcialmente	Se han conseguido adecuar los parámetros de contaminación lumínica para el laboratorio, pero a nivel de campo no fue posible.
OB-04	Sí	Se han adaptado los datos capturados y se han almacenado de forma correcta en la base de datos.
OB-05	Sí	Se han analizado los datos obtenidos consiguiendo prescindir de aquellos que son irrelevantess.

OB-06	Sí	Se ha conseguido desarrollar un modelo predictivo de contaminación lumínica basado en tecnologías de aprendizaje automático, consiguiendo mantener cierta modularidad y velocidad del sistema. Consiguiendo comparar distintos modelos.
OB-07	Sí	Se ha conseguido comprobar la validez del modelo, generando unas gráficas que ofrecen información sobre la validez del mismo.

Respecto al objetivo **OB-01**, se puede decir que se ha cumplido sin ningún problema, de hecho, en el apartado “**8.1. Elección de características**” se comenta de forma extensa todo este proceso.

Respecto al objetivo **OB-02**, ocurre algo similar al objetivo anterior, en el apartado “**8.2. Adquisición de los datos**” se comenta todo el proceso de adquisición de los datos y como se ha realizado, cumpliéndose así dicho objetivo. De hecho, dentro de este apartado, también se comenta como aparte de cómo se adquiere cada dato, como se almacenan cada patrón dentro del calc, para su posterior uso en un base de datos, por ende se cumple el objetivo **OB-04**.

Respecto al objetivo **OB-03**, se ha logrado a medias, ya que con el apartado “**8.2. Adquisición de los datos**”, se explica cómo se van obteniendo dichos datos a nivel de laboratorio, sin embargo, por falta de tiempo y la limitación del proyecto, no se ha logrado adaptar la adquisición de datos a nivel de campo.

Respecto al objetivo **OB-05**, se puede decir que se ha cumplido sin ningún problema, en el apartado “**8.3 Preprocesamiento de los datos y conclusiones sobre la elección de características**” se comenta de forma extensa el proceso de preprocesado estableciendo así que parámetros son determinantes.

Por último, tanto el objetivo **OB-06 y OB-07**, se han cumplido. En el apartado “**8.4. Creación del sistema de análisis y predicción de contaminación lumínica**”, se explica la creación y desarrollo del modelo para predecir la contaminación lumínica. También en este punto se presentan gráficas probando el funcionamiento de la misma.

Por lo tanto, podríamos llegar a la conclusión de que se ha conseguido llegar a lograr un modelo que es capaz de predecir la contaminación lumínica más o menos de forma correcta.

Esto lo que permite es que si queremos comprobar la contaminación lumínica que tiene una luminaria en la calle, solo con saber los datos de la luminaria y tomar la iluminancia inferior, ya podríamos obtener la contaminación lumínica de la misma, esto evitaría que tuviéramos que realizar una captura de los luxes superiores a la luminaria, lo cual sería complicado y dificultoso. Aparte, cómo calcular el flujo superior sería muy difícil porque se necesitaría un entorno muy controlado, con este modelo sería muy sencillo de obtener.

Finalmente, también permite comprobar si una luminaria en concreto cumple los requisitos para colocarse en ciertos tipos de E, únicamente se necesitaría obtener la iluminancia inferior (instalación de una única luminaria a cierta altura y tomando los luxes en el suelo) siendo más sencillo que instalar las luminarias y comprobando a posteriori.

Cabe destacar que el modelo no es perfecto y que tiene ciertos problemas, esto es lógico, ya que con una mayor cantidad de patrones y con una mayor riqueza de características se podría haber realizado un modelo mejor, pero con nos encontramos en un TFG donde hay que cumplir ciertas fechas, no se puede alargar el proyecto indefinidamente y se han optado por lo que se consideraban las mejores características y el número necesario de patrones para que el modelo funcionara de una forma que se considerara aceptable.

**Como mejoras del propio modelo a futuro**, se recomendaría de la inserción de mayor cantidad de patrones permitiéndole al modelo mayor precisión en su predicción y mayor riqueza en sus características probando mayor cantidad de luminarias con características diferentes e incluyendo nuevas como orientación, ópticas, etc.

Cabe destacar que el modelo se ha realizado con pruebas a nivel de laboratorio, en un futuro si se realiza el proyecto a nivel de campo, se recomendaría que se

tuvieran en cuenta más parámetros aún como la modificación de la reflexión del suelo en el caso de lluvia, el desgaste del pavimento, la reflexión de la luz al reflejarse en los coches y todos aquellos casos que no se tenían en cuenta dentro de un laboratorio.

Todo lo comentado anteriormente es a nivel de modelo, a nivel de sistema, personalmente, me encuentro satisfecho, ya que se ha conseguido que tanto el servicio donde se encuentra el modelo como la bd sea muy sencillo de utilizar con únicamente una sola línea de comandos. Aparte el uso de la api es relativamente sencillo y permite una gran modularidad para que otros programas puedan usarlo.

Finalmente solo me falta hablar de las **conclusiones a nivel personal**, personalmente me encuentro bastante satisfecho con el trabajo conseguido, lógicamente soy consciente de que siempre se puede llegar a lograr un poco más, pero los objetivos personales impuestos por mí han sido cumplidos. Pudiendo aprender de un campo que no disponía ningún conocimiento (el campo de la luminaria), aprendiendo el uso de api y docker cuando no lo conocía tan en profundidad como ahora y el hecho de ser capaz de sacar los datos del modelo a mano, de tal forma que he aprendido lo complejo que puede llegar a ser esto y lo laborioso de lo mismo. Permitiendo así valorar más la importancia de los datos incluso más que el propio modelo en sí.



## 11. Bibliografía

- ASPECTOS VIGENTES EN MATERIA DE PRESERVACIÓN DEL CIELO NOCTURNO EN ANDALUCÍA, TRAS LA ANULACIÓN DEL DECRETO. (n.d.). Junta de Andalucía. Retrieved July 23, 2022, from [https://www.juntadeandalucia.es/medioambiente/portal/documents/20151/812350/Normativa\\_cl\\_aspectos\\_vigentes.pdf/b3fcab3d-aa70-26da-5ed0-f3f143d14304?t=1516885052000](https://www.juntadeandalucia.es/medioambiente/portal/documents/20151/812350/Normativa_cl_aspectos_vigentes.pdf/b3fcab3d-aa70-26da-5ed0-f3f143d14304?t=1516885052000)
- Berzal, F. (2018). *Redes Neuronales and Deep Learning*. Fernando Berzal.
- Bishop, C. M., & Bishop, P. o. N. C. C. M. (2007). *Pattern Recognition and Machine Learning* (Springer Science+Business Media, Ed.). Springer.
- BOE-A-2008-18634 Real Decreto 1890/2008, de 14 de noviembre, por el que se aprueba el Reglamento de eficiencia energética en instalaciones de alumbrado exterior y sus Instrucciones técnicas complementarias EA-01 a EA-07. (2021, November 14). BOE.es. Retrieved July 2, 2022, from [https://www.boe.es/diario\\_boe/txt.php?id=BOE-A-2008-18634](https://www.boe.es/diario_boe/txt.php?id=BOE-A-2008-18634)
- Castaneyra, A. (2021, February 24). *MK350S Premium Spectrometer*. UPRtek. Retrieved July 9, 2022, from <https://www.uprtek.com/es/product/spectrometers/mk350s-premium>
- Contaminación lumínica – Stars4All. (2015). Stars4All. Retrieved July 2, 2022, from <https://stars4all.eu/light-pollution-2/?lang=es>
- Docker. (n.d.). Docker: Home. Retrieved July 9, 2022, from <https://www.docker.com/>

- Donners, M. (2018, Octubre). Colors of attraction: Modeling insect flight to light behavior.

*JOURNAL OF EXPERIMENTAL ZOOLOGY PART A-ECOLOGICAL AND  
INTEGRATIVE PHYSIOLOGY*, 434-440. 10.1002/jez.2188

- *Fundamentos sobre la generación de la luz y el alumbrado*. (2011, Septiembre). Philips.

Retrieved July 9, 2022, from

<https://www.assets.signify.com/is/content/PhilipsConsumer/PDFDownloads/Spain/Basics-of-lighting.pdf>

- *Guía técnica de adaptación de las instalaciones de alumbrado exterior al decreto 375/2010 de 3 de agosto*. (2013, November 15). YouTube. Retrieved July 9, 2022, from  
<https://www.famp.es/export/sites/famp/.galleries/documentos-lab-eficiencia-energetica-a-guias/GUIA-11.pdf>

- GUTIERREZ, F. J. (n.d.). *MEMORIA JUSTIFICATIVA DEL PROYECTO DE DECRETO POR EL QUE SE REGULA LA COMPOSICIÓN Y FUNCIONES DE LA COMISIÓN DE EVALUACIÓN Y*. Junta de Andalucía. Retrieved July 2, 2022, from  
[https://www.juntadeandalucia.es/sites/default/files/2022-02/Memoria%20Justificativa\\_0.pdf](https://www.juntadeandalucia.es/sites/default/files/2022-02/Memoria%20Justificativa_0.pdf)

- *Inicio Índice espectral G*. (n.d.). Junta de Andalucía. Retrieved July 2, 2022, from  
[https://www.juntadeandalucia.es/medioambiente/portal/landing-page/-/asset\\_publisher/4V1kD5gLiJkq/content/-c3-adndice-espectral-g/20151](https://www.juntadeandalucia.es/medioambiente/portal/landing-page/-/asset_publisher/4V1kD5gLiJkq/content/-c3-adndice-espectral-g/20151)

- John Shawe-Taylor, & Nello Cristianini. (23 marzo 2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods* (1st Edición ed.). Cambridge University Press.

- *MATLAB - El lenguaje del cálculo técnico - MATLAB & Simulink*. (n.d.). MathWorks.  
Retrieved July 2, 2022, from <https://es.mathworks.com/products/matlab.html>

- Olsen, R. N. (2014, JUN 3). Modelling US light pollution. *JOURNAL OF ENVIRONMENTAL PLANNING AND MANAGEMENT*, 883-903. 10.1080/09640568.2013.774268
- Pal, C. J., Hall, M. A., Witten, I. H., & Frank, E. (2016). *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier Science.
- Python. (n.d.). Welcome to Python.org. Retrieved July 9, 2022, from  
<https://www.python.org/>
- scikit. (n.d.). scikit-learn: machine learning in Python — scikit-learn 1.1.1 documentation. Retrieved July 9, 2022, from <https://scikit-learn.org/stable/>
- scikit-learn developers (BSD License). (2007). *sklearn.ensemble.RandomForestRegressor — scikit-learn 1.1.2 documentation*. Scikit-learn. Retrieved September 7, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html?highlight=randomforest#sklearn.ensemble.RandomForestRegressor>
- scikit-learn developers (BSD License). (2007). *sklearn.linear\_model.LinearRegression — scikit-learn 1.1.2 documentation*. Scikit-learn. Retrieved September 7, 2022, from [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LinearRegression.html?highlight=linearregression#sklearn.linear\\_model.LinearRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linearregression#sklearn.linear_model.LinearRegression)
- scikit-learn developers (BSD License). (2007). *sklearn.linear\_model.SGDRegressor — scikit-learn 1.1.2 documentation*. Scikit-learn. Retrieved September 7, 2022, from [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.SGDRegressor.html?highlight=sgdregressor#sklearn.linear\\_model.SGDRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html?highlight=sgdregressor#sklearn.linear_model.SGDRegressor)
- scikit-learn developers (BSD License). (2007). *sklearn.svm.SVR — scikit-learn 1.1.2 documentation*. Scikit-learn. Retrieved September 7, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html?highlight=svr#sklearn.svm.SVR>

- scikit-learn developers (BSD License). (2007). *sklearn.tree.DecisionTreeRegressor—scikit-learn 1.1.2 documentation*. Scikit-learn.  
<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html?highlight=decisiontree#sklearn.tree.DecisionTreeRegressor>
- Tan, Pang-Ning. (1 enero 2005). *Introduction to Data Mining* (N.º 1 edición ed.). Addison Wesley.
- Witten, I. H., Hall, M. A., Pal, C. J., & Frank, E. (2016). *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier Science.



## Agradecimientos

A mi **familia** por apoyarme siempre durante esta etapa en todo lo necesario.

A **Ezequiel Herruzo** por darme esta idea y la oportunidad de trabajar en un proyecto tan interesante.

A **Fco. Ramón Lara** por ayudarme siempre que tenía algún problema con el material o otros recursos que necesitaba o con conceptos que no entendía.

A **Eduardo Ruiz** por aportar su punto de vista sobre el proyecto y ofrecer distintas luminarias para probar en el laboratorio. También por la explicación de las fórmulas para poder así obtener la contaminación lumínica.

A **Álvaro David Domínguez** por ayudar con los cables para poder enchufar las luminarias a la corriente permitiendo agilizar el trabajo a la hora de tomar los datos.

A **Rafael Hormigo** por ayudarme en el montaje de la estructura, ya que sin su ayuda no habría sido posible. También por ayudarme con ciertos problemas o dudas a la hora de realizar el modelo y el docker. Por último, con la compra de algunos materiales para la realización del proyecto.



# **Anexos**

Anexo I. Manual de usuario.

Anexo II. Manual de código.

## **ANEXO I**

## **MANUAL DE USUARIO**

## Anexo I. Manual de usuario.

Con el manual de usuario se pretende explicar de forma clara y concisa la instalación y del sistema y el contenido del mismo, con una breve explicación de su correcto uso.

### 1. Instalación:

En este apartado se explicará la instalación del sistema y de todos los requerimientos necesarios.

#### 1.1. Instalación Docker:

Primeramente es necesario instalar **Docker** en el dispositivo, para ello se recomienda acceder a la página de docker y seguir sus pasos para la instalación. Si por ejemplo quisiéramos instalar **Docker** en una máquina con distribución ubuntu, debemos seguir la siguiente guía:

-<https://docs.docker.com/engine/install/ubuntu/>

En el enlace anterior podemos encontrar una guía específica para ubuntu, pero en la propia página de **docker**, hay guías para windows u otras distribuciones por si fuera necesario su instalación.

Con el paso anterior instalaremos **docker**, pero para mayor comodidad instalaremos también **docker compose**. Esto permitirá que si disponemos de un **docker compose** creado (el sistema dispone de uno), no sea necesario lanzar cada contenedor por separado con sus dependencias, permitiendo una instalación más sencilla.

Igual que en el caso anterior, para instalar **docker-compose** seguiremos los pasos dados en la propia página de **docker**, dependiendo de la distribución usada. El enlace sería el siguiente:

-<https://docs.docker.com/compose/install/>

En el caso de ubuntu, se usaría, el mismo enlace que en el primer caso ya que en el apartado “Install Docker Engine”, se instala **Docker** con **docker-compose**.

Una vez realizado los pasos necesarios, ya tendríamos **docker** y **docker-compose** para poder instalar el sistema.

## 1.2. Descarga de github

Una vez instalado el **docker** y el **docker engine**, necesitamos descargarnos el sistema. Para ello accederemos a la página donde se encuentra el proyecto en **github**, siendo la siguiente:

[https://github.com/Witiza99/TFG\\_SISTEMA\\_DE\\_ANALISIS\\_Y\\_PREDICCI-N\\_DE\\_CONTAMINACION\\_LUMINICA](https://github.com/Witiza99/TFG_SISTEMA_DE_ANALISIS_Y_PREDICCI-N_DE_CONTAMINACION_LUMINICA)

Una vez accedemos a dicha página, podremos directamente descargarnos el archivo en un **.zip** o copiar el comando que aparece en el botón clone para, al introducirlo en un terminal, copiar y enlazar el proyecto al **github**. En la siguiente imagen se puede ver el comando y donde se obtiene:

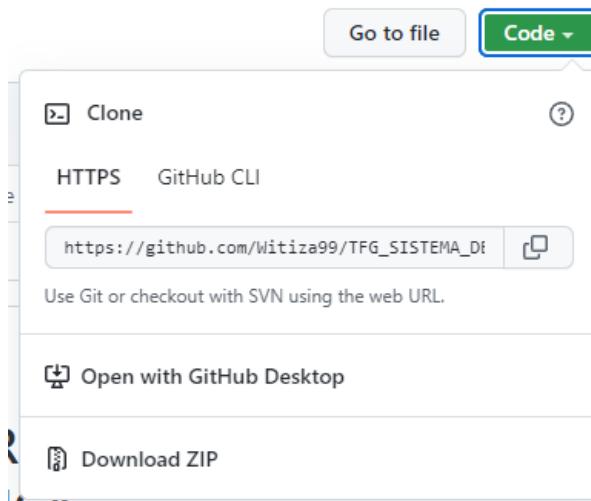


Imagen Anexo (Manual Usuario) 1.2.1: comando git clone

Si todo ha salido correctamente, deberíamos de tener una carpeta donde dentro se encuentra todo el sistema. Una vez realizados todos los pasos anteriores, ya tendríamos todo lo necesario instalado para poder encender el sistema.

### 1.3. Descargas auxiliares para programas de apoyo

Al usar docker compose no será necesario realizar los imports y descargas a mano ya que se realizan de forma automática.

Al lanzar docker compose up, este comando lanzará los dos servicios, creando la imagen necesaria y construyendo todos los contenedores con las dependencias necesarias (python, numpy, scikit, etc). El sistema como tal, ya se encontrará lanzado y preparado para su uso.

En el caso de querer usar un programa que no sea el sistema, es decir, los programas de apoyo para **generar gráficas, preprocessamiento o incluso el programa que simula un cliente**, será necesario entonces instalar en la propia máquina **python** y las dependencias que necesite dicho programa.

Dependiendo de la distribución es diferente, en el caso de ubuntu sería sudo **apt install** y la **versión de python**. Para las dependencias, **sudo install pip** y el **nombre del paquete a instalar**, en la propia cmd al intentar ejecutar el programa se comenta el nombre de las dependencias que faltan.

## 2. Explicación de las carpetas

Una vez accedemos a la carpeta descargada, nos deberíamos de encontrar lo siguiente:



Imagen Anexo (Manual Usuario) 2.1: directorio del sistema

Nos encontramos cuatro carpetas y dos ficheros. Primero se van a explicar qué son estos dos ficheros:

El fichero **docker-compose.yaml** es el fichero que nos va a permitir arrancar el sistema con sus respectivos contenedores (se explica con más profundidad en el manual de código) y el fichero **DatosLuminaria.ods**, es un fichero que contiene la primera versión de los datos adquiridos sin realizar ninguna modificación sobre los mismos.

Respecto a las carpetas que nos encontramos:

- **base\_de\_datos**: esta carpeta va a contener toda la información necesaria del contenedor creado por la imagen *mysql*. Dentro podemos encontrar lo siguiente:



Imagen Anexo (Manual Usuario) 2.2: directorio de base\_de\_datos

La carpeta **database** contiene la información necesaria del contenedor, como se comentó anteriormente. El fichero **README\_EstructuraBD.txt** es un fichero auxiliar para entender mejor la estructura de la base de datos. El contenido del fichero es el siguiente:

```

1  Este fichero contiene información sobre el esquema de la base de datos.
2  -----
3
4
5  1. Tabla de datos_luminaria;
6
7  Esta tabla contiene información sobre los atributos extraídos para los experimentos.
8
9  - Variables:
10
11  + Nombre de luminaria usada en ese experimento
12  + Altura donde se coloca la luminaria en el experimento (cm)
13  + Flujo Luminico de la luminaria colocada (lm)
14  + TCC, Temperatura Correlada del Color (K)
15  + Iluminancia detectada a nivel de suelo (lux)
16  + Espectro producido por la luminaria(valor maximo nm)
17  + Color del suelo usado en el experimento
18  + Indice de reflectancia para el suelo puesto
19  + Iluminancia en el punto justo superior a la luminaria (lux)
20  + Flujo Luminico superior a la luminaria (lm)
21  + FHSI, Flujo Hemisferio Superior instalado, porcentaje de flujo que se desprende
22  por encima de la luminaria dependiendo del flujo total(%)
23
24
25
26      Field          |      Type       | Null | Key | Default | Extra
27  -----
28  TipoLuminaria   | varchar(255) | YES  |     |         |
29  AlturaLuminaria | float        | YES  |     |         |
30  FlujoLuminicoTotal | float      | YES  |     |         |
31  TCC             | float        | YES  |     |         |
32  IluminanciaAbajo | float        | YES  |     |         |
33  Espectro         | float        | YES  |     |         |
34  ColorSuelo       | varchar(255) | YES  |     |         |
35  ReflectanciaSuelo | float      | YES  |     |         |
36  IluminanciaSuperior | float      | YES  |     |         |
37  FlujoSuperior    | float        | YES  |     |         |
38  FHSI            | float        | YES  |     |         |

```

Imagen Anexo (Manual Usuario) 2.3: REAME\_de\_la\_bd

- **modelo\_prediccion\_contaminacion\_luminica:** esta carpeta va a contener toda la información necesaria del contenedor creado por la imagen *modelo\_prediccion\_contaminacion\_luminica*. Dentro podemos encontrar lo siguiente:

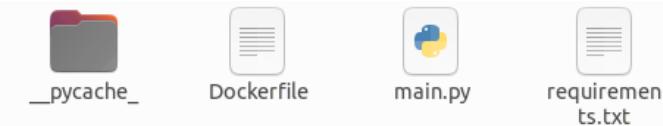


Imagen Anexo (Manual Usuario) 2.4: directorio\_de\_modelo\_prediccion\_contaminacion\_luminica  
 Los ficheros **Dockerfile** y **requirements.txt** son ficheros que se utilizan a la hora de generar el contenedor (se explicará el contenido en el manual de código). La carpeta **\_\_pycache\_\_** es una carpeta que genera el docker para sincronizar el **main.py** de este directorio con la copia que se genera dentro del contenedor.

El **main.py** es el código principal, donde podemos encontrar el código de todas las peticiones *get* posibles.

Una vez se realice el entrenamiento, aparecerán nuevos ficheros, estos ficheros son archivos **.pickle** y **ficheros con los nombres de los modelos**. Los archivos **pickle** los podemos ignorar ya que son los modelos ya entrenados. Los ficheros con nombres de modelos son los resultados obtenidos con su **MSE** con distintas combinaciones de parámetros, en la última línea encontramos el mejor resultado. Se recalca este dato, ya que puede llegar a ser interesante ver los resultados de los entrenamientos.

- **preprocesamiento:** esta carpeta contiene el programa auxiliar creado para el procesamiento de los datos (histogramas y mapa de correlaciones). Dentro podemos encontrar tanto el programa como los datos que se preprocesan:



Imagen Anexo (Manual Usuario) 2.5: directorio\_preprocesamiento

- **cliente:** esta carpeta contiene el programa auxiliar creado para simular un cliente y usar de forma más sencilla el sistema sin realizar *get* de predicción directamente por navegador. También contiene el código para generar gráficas comparativas entre los resultados esperados y los obtenidos. La imagen del directorio es la siguiente:

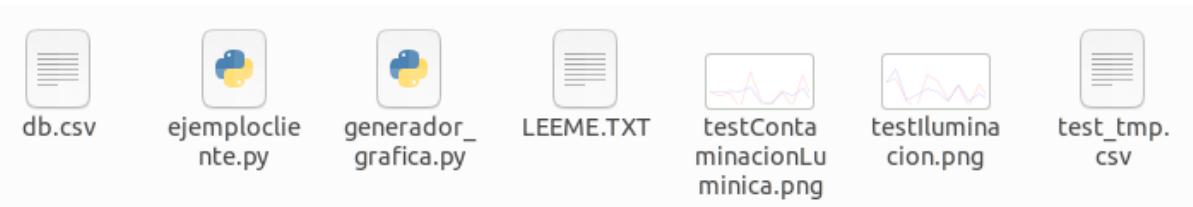


Imagen Anexo (Manual Usuario) 2.6: directorio\_preprocesamiento

Cabe destacar que el fichero **LEEME.TXT** explica brevemente los dos .py y el uso del **generador\_grafica.py**

### 3. Uso de docker

Una vez se ha explicado la instalación del sistema y que es cada cosa. Podemos arrancar el sistema. Para arrancar el sistema es necesario lanzar el docker compose, para que funcionen los contenedores.

Para ello se abre una terminal en el directorio donde está el **docker-compose.yaml** y se introduce “**(sudo)<sup>1</sup> docker compose up**”.

Para parar el sistema, se debe lanzar un “ctrl-C” y esperar que se cierre correctamente, si aun así no se detiene el sistema, sería necesario enviar otro “ctrl-C”, pero no se recomienda ya que se puede producir algún fallo en la base de datos por ejemplo.

Si queremos borrar los contenedores (pero no las imágenes) podemos introducir el comando “**(sudo) docker compose down**”.

Si se modifica el **Dockerfile** para añadir algún tipo de *import* por algún motivo, es necesario actualizar la imagen, para eso es necesario eliminar tanto los contenedores de esa imagen como la propia imagen, para que se actualice.

Para eliminar una imagen sería “**(sudo) docker image rm <id imagen>**” (para ver el id de todas las imágenes y así elegir la que queramos “**(sudo) docker images**”)

---

<sup>1</sup> Sudo solo es necesario si no se tienen configurados los permisos para el usuario Docker.

## 4. Peticiones get

Tras ver un poco el funcionamiento de docker y arrancar el sistema con “**(sudo) docker compose up**”, podemos realizar peticiones *get* al sistema.

El contenedor **modelo\_prediccion\_contaminacion\_luminica** cuenta con varias peticiones; insertar, entrenar y predecir. La estructura general de una petición es la siguiente: GET {ip}:{puerto}/{nombre\_peticion\_get}/{parámetros}.

A continuación se explica cada petición *get* que nos podemos encontrar:

### 4.1. Petición: insertar patrones

Esta petición realiza la inserción de los patrones pasados en un fichero a la base de datos especificada en la petición.

Los parámetros de esta función son:

- **base\_de\_datos**: le pasamos el nombre de la base de datos a utilizar, la base de datos debe contener una tabla llamada **datos\_luminaria** para que esta función trabaje correctamente.
- **archivo\_datos**: le pasamos el nombre del fichero donde se encuentran los patrones, debe estar en formato csv para que funcione correctamente.

Esta función devuelve un mensaje de éxito o error.

### 4.2. Petición: entrenar

Esta función nos permite entrenar el modelo utilizando todos los algoritmos disponibles, nos genera tres archivos para cada algoritmo, uno con los resultados del entrenamiento, otro con el algoritmo guardado con los mejores parámetros y el scaler de dicho modelo, para así poder utilizarlo más tarde.

Los parámetros de esta función son:

- **base\_de\_datos**: le pasamos el nombre de la base de datos a utilizar, la base de datos debe contener una tabla llamada **datos\_luminaria** para que esta función trabaje correctamente.
- **crear\_nuevo\_fichero\_entrenamiento**: este parámetro es opcional, indica si se desea entrenar con el fichero de datos ya generado o si se actualiza con el

de la base de datos (esta flag solo se usa para el generador de gráficas). Es un bool y su valor por defecto es True.

Esta función devuelve un mensaje de éxito o error.

#### 4.3. Función: predecir

Esta función permite predecir la contaminación lumínica (prediciendo la iluminancia superior) para un patrón recibido. Para el correcto funcionamiento de esta función se necesita haber entrenado previamente.

Los parámetros que recibe esta función son:

- **ColorSuelo**: parámetro que indica el color del suelo.
- **AlturaLuminaria**: parámetro que indica la altura de la luminaria.
- **FlujoLuminicoTotal**: parámetro que indica el flujo lumínico total de la luminaria.
- **TCC**: parámetro que indica la Temperatura Correlada del Color de la luminaria.
- **IluminanciaAbajo**: parámetro que indica la iluminancia en el punto inferior.
- **Espectro**: parámetro que indica el espectro de la luz emitida por la luminaria.
- **ReflectanciaSuelo**: parámetro que indica la reflectancia del suelo.
- **IluminanciaSuperior**: parámetro que indica la iluminancia en el punto superior a la luminaria.

Esta función devuelve un JSON con la iluminancia superior predicha.

## 5. Uso del sistema

Ya visto y explicado todos los puntos anteriores, se realiza un ejemplo de uso del sistema.

Primeramente el sistema necesita tener datos en la base de datos, con los que poder entrenar. Una vez dispone de los datos, se realiza el entrenamiento y finalmente, si se tiene el modelo ya entrenado, se realiza la predicción. Por lo tanto, teniendo en cuenta este orden, primero se realiza la inserción de los datos. (Todo esto es posible si el sistema se encuentra encendido con **sudo docker-compose up**)

Para realizar la inserción de los datos, necesitamos abrir un navegador y acceder a dicho servicio. Para ello, especificamos el puerto y dirección donde se encuentra dicho servicio y para que funcione correctamente, le pasamos como argumentos la base de datos donde queremos insertar los patrones y el fichero que queremos insertar (el fichero se debe encontrar dentro del directorio del modelo). Con la siguiente imagen se comprende mejor:



Imagen Anexo (Manual Usuario) 5.1: navegador\_insertarPatrones

Entonces introduciremos la siguiente dirección al navegador:

[http://localhost:8000/insertarPatrones/?base\\_de\\_datos=db&archivo\\_datos=DatosLuminaria.csv](http://localhost:8000/insertarPatrones/?base_de_datos=db&archivo_datos=DatosLuminaria.csv)

Si la petición fue correcta, en el navegador nos saldrá un mensaje de éxito.

Si los datos ya se encuentran en la base de datos, podremos realizar el entrenamiento. El proceso para llamar al entrenamiento y la predicción es similar.

Para realizar el entrenamiento introduciremos:

[http://localhost:8000/entrenar/?base\\_de\\_datos=db&crear\\_nuevo\\_fichero\\_entrenamiento=True](http://localhost:8000/entrenar/?base_de_datos=db&crear_nuevo_fichero_entrenamiento=True)

En este caso, cuando se realice el entrenamiento, nos generará dentro de la carpeta del modelo, dentro de app, los siguientes **archivos .pickle y ficheros con los nombres de los modelos**.

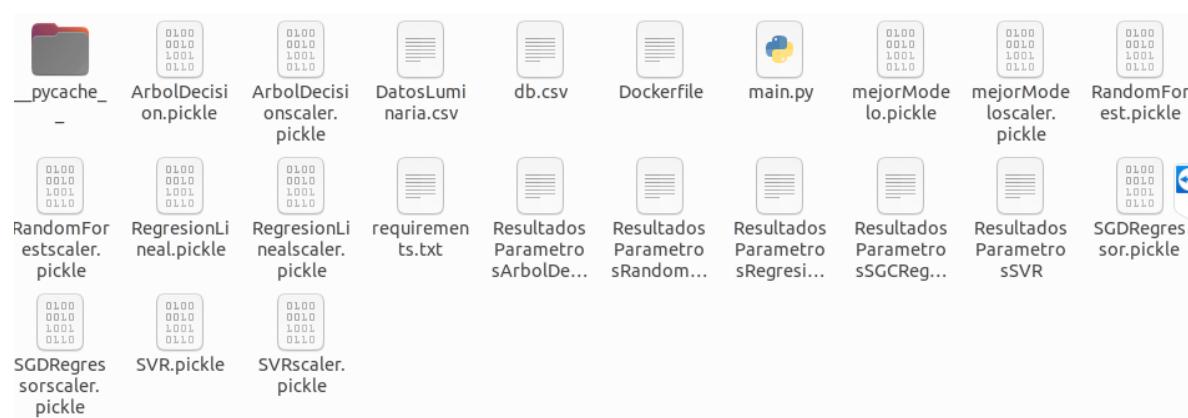


Imagen Anexo (Manual Usuario) 5.2: directorio\_de\_modelo\_prediccion\_contaminacion\_luminica\_tras\_entrenamiento

Si disponemos del mejor modelo, ya se puede predecir todas las veces que se deseé, para ello, se debería introducir:

<http://localhost:8000/predecir/?ColorSuelo=rojo&AlturaLuminaria=160&FlujoLuminicoTotal=3000&TCC=4000&IluminanciaAbajo=1600&Espectro=432&ReflectanciaSuelo=46&IluminanciaSuperior=0>

El caso anterior es un ejemplo de patrón, se puede introducir cualquier patrón mientras sea correcto. Se recomienda que para realizar predicciones se use el programa de ejemplo de cliente ya que es mucho más intuitivo y sencillo.

Si lanzamos el programa cliente (para ello es necesario instalar python y las dependencias necesarias explicadas en el apartado de instalación), con **python3.8 ejemplocliente.py** (estando en el directorio donde se encuentra dicho archivo), nos pedirá introducir cada dato de una forma más amigable que introduciendo directamente el patrón al navegador. En la siguiente imagen se puede apreciar:

```
antonioogg@antonioogg-SATELLITE-L50D-C:~/Escritorio/cosas_TFG/ProyectoTFG/SISTEMA_DE_ANALISIS_Y_PREDICCION_DE_CONTAMINACION_LUMINICA/cliente$ python3.8 ejemplocliente.py
Introduzca ColorSuelo:
1.Negro
2.GrisOscuro
3.GrisClaro
4.Blanco
5.Morado
6.Rojo
7.VerdeClaro
8.Verde
9.VerdeOscuro
10.MarronClaro
11.MarronOscuro
```

Imagen Anexo (Manual Usuario) 5.3: entrada\_de\_parametros\_cliente

Tras introducir todos los datos, se realizará la petición y se mostrarán los resultados obtenidos para dicho patrón. En la siguiente imagen se pueden ver los resultados:

```
La IluminanciaSuperior es : 51.332
La FlujoSuperior es: 96.248
La contaminación Luminica es: 3.208 %
Como la contaminacion luminica es superior al 1%, dicha luminaria solo podria usarse en el tipo de areas
luminicas E4, E3 y E2
```

Imagen Anexo (Manual Usuario) 5.4: salida\_cliente

Aparte se calcula el flujo lumínico y la contaminación lumínica para facilitar aún más el entendimiento y se explica en qué tipos de e se podría situar una luminaria con dichas características.



## **ANEXO II**

## **MANUAL DE CÓDIGO**

## Anexo II. Manual de código.

Se adjunta en el siguiente enlace el github donde se encuentra alojado todo el código del proyecto:

[https://github.com/Witiza99/TFG\\_SISTEMA\\_DE\\_ANALISIS\\_Y\\_PREDICCI-N\\_DE\\_C\\_ONTAMINACION\\_LUMINICA](https://github.com/Witiza99/TFG_SISTEMA_DE_ANALISIS_Y_PREDICCI-N_DE_C_ONTAMINACION_LUMINICA)

Se procede a explicar brevemente el código realizado en este proyecto.

### 1. Docker Compose

Para arrancar y configurar los docker, nos encontramos un archivo llamado **docker-compose.yaml**. En este archivo encontramos las especificaciones de las dos imágenes (**mysql** y **modelo\_prediccion\_contaminacion\_luminica**), así como los volúmenes que se le asignan a cada servicio y las dependencias existentes entre imágenes. Con la ayuda de la siguiente imagen se explicará la configuración de dicho archivo.

```
1  version: "3.3"
2
3
4  services:
5
6    modelo_prediccion_contaminacion_luminica:
7      build: ./modelo_prediccion_contaminacion_luminica/app
8      image: modelo_prediccion_contaminacion_luminica
9      ports:
10        - 8000:8000
11      stdin_open: true
12      tty: true
13      volumes:
14        - ./modelo_prediccion_contaminacion_luminica/app:/usr/src/app
15      depends_on:
16        - mysql
17
18    mysql:
19      image: mysql:latest
20      ports:
21        - 3306:3306
22      environment:
23        MYSQL_DATABASE: db
24        MYSQL_USER: user
25        MYSQL_PASSWORD: password
26        MYSQL_ROOT_PASSWORD: password
27      volumes:
28        - ./base_de_datos/database:/var/lib/mysql
```

Imagen Anexo (Manual Código) 1: docker-compose.yaml

Como podemos observar en la imagen 1, primeramente se especifica la versión de docker que se va a utilizar, en este caso la versión 3.3. En la línea 4, se especifican los servicios, siendo cada imagen que se va a utilizar, añadiendo un nombre exclusivo para cada imagen. Para nuestro caso encontramos dos servicios con una imagen cada una. El primer servicio es el sistema con los modelos llamado **modelo\_prediccion\_contaminacion\_luminica**, coincidiendo así con el nombre de la imagen. El segundo servicio es el servicio **mysql**, coincidiendo con el nombre de la imagen pero con la diferencia de que le añadimos “:latest”.

Primeramente se explicará el servicio **modelo\_prediccion\_contaminacion\_luminica**, explicando los diferentes parámetros que nos vamos a encontrar:

- **build**: este parámetro indica el directorio donde se encuentra el archivo Dockerfile que configura la imagen.
- **image**: aquí se indica la imagen a utilizar para la construcción del **contenedor del Docker**, puede ser una imagen predefinida que se descarga de internet o como en este caso, el nombre de la imagen que más tarde se especifica en el archivo **Dockerfile**. Este será el nombre que se le asigne al servicio
- **ports**: este parámetro especifica el puerto que se utilizará tanto en el contenedor del **Docker** como en la propia máquina donde se despliega el sistema.
- **stdin\_open**: equivalente al comando “docker run -i”, con este comando se mantiene el STDIN abierto.
- **tty**: equivalente al comando “docker run -t”, este comando asegura que se mantiene una consola abierta en nuestra máquina, equivalente a una consola dentro de la imagen.
- **volumes**: es un parámetro donde se especifica primero la carpeta de nuestra máquina donde instalar el volumen y seguido de los dos puntos, se le especifica la carpeta dentro del contenedor de **Docker** a donde queremos que se suban los archivos de la máquina donde se están desplegando los servicios.
- **depends\_on**: con este parámetro se asignan las dependencias necesarias para el servicio, de tal forma que dicho servicio no arranca hasta que arrancan primero los otros servicios de los cuales depende.

Se procede a analizar el segundo servicio, siendo mysql.

Como se puede ver en la imagen 1, podemos ver que los parámetros escogidos para configurar este servicio son ligeramente diferentes, esto se debe a que se usa una imagen descargada del propio docker, en vez de crearla directamente. La imagen que se especifica en el parámetro **image**, se busca a través de internet y se descarga automáticamente, una vez se lance el archivo docker-compose.

En el parámetro **environment** se especifican las variables necesarias para que funcione correctamente la base de datos creada por este servicio, en este caso se especifica el nombre de la base de datos, el usuario y su contraseña y finalmente, la contraseña de ROOT.

También se encuentran los parámetros de puerto y volumen que se explicaron anteriormente.

## 2. Dockerfile

Como vimos en el punto anterior, las imágenes pueden ser creadas por uno mismo, o descargadas por internet. Para nuestro caso se creó una imagen personalizada, para el primer servicio, siendo la siguiente:

```
1  FROM python:3.8
2  COPY . /usr/src/app
3  WORKDIR /usr/src/app
4
5  RUN pip install -r requirements.txt
6  RUN pip install numpy
7  RUN pip install pandas
8  RUN pip install sklearn
9  RUN pip install scipy
10 RUN pip install mysql-connector-python
11
12
13 ENTRYPOINT uvicorn --host 0.0.0.0 main:app --reload --port 8000
14
```

Imagen Anexo (Manual Código) 2: Dockerfile

Como se puede observar en la imagen 2, esta sería la configuración de la imagen que genera el servicio **modelo\_prediccion\_contaminacion\_luminica** que coincide con el nombre de la imagen. Primeramente, en la configuración se especifica la versión de python que se pretende utilizar y se copia todo el contenido del directorio donde se encuentra este fichero (debe encontrarse junto con el main.py) al directorio especificado “/usr/src/app” y que defina dicho directorio como espacio de trabajo. Este proceso se realiza para que al actualizar el fichero main.py se apliquen los cambios dentro del fichero copia que se genera dentro del propio contenedor.

A continuación, se utiliza el comando **RUN** como si fuera una comando de terminal, de tal forma que junto con pip se instalan aquellas dependencias que se vayan a utilizar dentro del modelo como numpy, pandas, sklearn, etc.

Cabe destacar que se añadieron las dependencias de **fastapi** y **uvicorn** a través del fichero requirements.txt, se puede observar mejor en la siguiente imagen:

```
1 fastapi==0.59.0
2 uvicorn==0.11.5
```

Imagen Anexo (Manual Código) 3: Requirements.txt

Por último, falta comentar el comando **ENTRYPOINT**. Con este comando se definen los parámetros del servidor **uvicorn**, de tal forma que, con **--host** se especifica la ip del servidor, en este caso 0.0.0.0, es decir localhost; con **main:app** se especifica la carpeta de la aplicación, que en este caso es app; con **--reload** se consigue que si se realiza alguna modificación en el fichero **Python** se recargue la aplicación; finalmente con **--port** se especifica el puerto en el que se ejecutará la aplicación.

### 3. main.py

Este fichero se encuentra dentro del servicio **modelo\_prediccion\_contaminacion\_luminica**, y contiene el programa principal que utiliza principalmente el sistema. En este apartado se explicará el código del mismo, diferenciando así las peticiones “get” de las funciones auxiliares.

```
1  #paquetes necesarios para el funcionamiento del programa
2  from fastapi import FastAPI
3  from fastapi.responses import JSONResponse
4  from fastapi.encoders import jsonable_encoder
5  import numpy as np
6  import pandas as pd
7  from typing import Optional
8
9  import mysql.connector #paquete para insertar en la bd de mysql
10
11 from sklearn import svm
12 from sklearn import linear_model
13 from sklearn import tree
14 from sklearn import ensemble
15 from sklearn import preprocessing
16 from sklearn.model_selection import train_test_split
17 from sklearn.preprocessing import MinMaxScaler, StandardScaler
18 from sklearn.compose import ColumnTransformer
19 from sklearn.preprocessing import OneHotEncoder
20 from sklearn.metrics import mean_squared_error
21 from sklearn.metrics import mean_absolute_error
22
23 #import requests
24 import json
25 import pickle
26
```

Imagen Anexo (Manual Código) 4: import main.py

Como podemos ver en la imagen 4, primeramente se importan al programa todas aquellas dependencias que se vayan a utilizar durante el código. Estas van desde el módulo de fastapi para poder usar peticiones “get”, hasta métodos de sklearn.

Primeramente, se explican las peticiones get:

```
27
28 #####PETICIONES GET#####
29 #####
30 #####
31
32 app = FastAPI()
33
34 @app.get("/")
35 def read_root():
36     return {"No se realiza ninguna acción"}
37
```

Imagen Anexo (Manual Código) 5: get vacío main.py

Como podemos observar en la imagen 5, el primer get es un get vacío, que se creó de forma auxiliar, para aquellos casos donde el usuario no especifica en su petición ningún tipo de get de los existentes.

```

39  @app.get("/insertarPatrones")#funcion para insertar datos en la bd
40  def read_root(base_de_datos: str, archivo_datos: str):
41
42      #guardamos los datos en un dataset de tipo string
43      try:
44          Dataset = pd.read_csv(archivo_datos)
45          #cambio a float aquellos datos guardados en int64
46          Dataset['ReflectanciaSuelo'] = Dataset['ReflectanciaSuelo'].astype('float64')
47          Dataset['AlturaLuminaria'] = Dataset['AlturaLuminaria'].astype('float64')
48          Dataset['FlujoLuminicoTotal'] = Dataset['FlujoLuminicoTotal'].astype('float64')
49          Dataset['TCC'] = Dataset['TCC'].astype('float64')
50          Dataset['Espectro'] = Dataset['Espectro'].astype('float64')
51      except:
52          return {"Error con el fichero de datos introducido"}
53      print(Dataset)
54
55      #Creamos conexion con la bd
56      conn = mysql.connector.connect(
57          user='root', password='password', host='mysql', port=3306, database=base_de_datos)
58
59      #Creamos un cursor
60      cursor = conn.cursor()
61
62      for i in Dataset.index:#recorremos toda la database para almacenarla en la bd
63          # Preparamos una solicitud SQL para insertar en la bd.
64          sql = """INSERT INTO datos_luminaria(
65              TipoLuminaria, AlturaLuminaria, FlujoLuminicoTotal, TCC, IluminanciaAbajo, Espectro,
66              ColorSuelo, ReflectanciaSuelo, IluminanciaSuperior, FlujoSuperior, FHSI)
67              VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""

```

Imagen Anexo (Manual Código) 6: get insertar main.py primera parte

```

68
69      try:
70          # Se ejecuta el comando
71          cursor.execute(sql, (Dataset["TipoLuminaria"][i], Dataset["AlturaLuminaria"][i], Dataset["FlujoLuminicoTotal"][i],
72          Dataset["TCC"][i], Dataset["IluminanciaAbajo"][i], Dataset["Espectro"][i],
73          Dataset["ColorSuelo"][i], Dataset["ReflectanciaSuelo"][i], Dataset["IluminanciaSuperior"][i],
74          Dataset["FlujoSuperior"][i], Dataset["FHSI"][i]))
75
76          conn.commit()#guardamos los cambios
77      except:
78          # Rollback en caso de error con la insercion
79          conn.rollback()
80
81      # Se cierra la conexion
82      conn.close()
83
84      return {"Exito en la insercion de patrones"}

```

Imagen Anexo (Manual Código) 7: get insertar main.py segunda parte

Tanto en la imagen 6 como 7 encontramos el “get” de insertar patrones. Primeramente, se lee del fichero csv (pasado el nombre del fichero a través de la petición get), y se almacena en un Dataset. Una vez almacenados todos los patrones en el Dataset, se realiza la conexión con la base de datos y se realiza una petición de inserción tantas veces como patrones se hayan almacenado en el Dataset. Si todo ha salido correctamente, se cierra la conexión y da un mensaje de éxito.

```

88 @app.get("/entrenar")#get que entrena el modelo
89 def read_root(base_de_datos: str, crear_nuevo_fichero_entrenamiento: Optional[bool] = True):
90
91     #creamos el fichero para entrenar con los datos que deseamos de la base de datos
92     if crear_nuevo_fichero_entrenamiento:
93         if '-I' == crear_fichero_entrenamiento(base_de_datos):#funcion que crea el fichero con los datos de la bd
94             return {"Error al crear el archivo con los patrones de la bd"}
95
96     fichero_entrenamiento = "db.csv"
97
98     #llamamos a funcion read data para extraer los datos del fichero csv a un formato con el que podamos trabajar
99     Dataset, train_inputs, train_outputs, test_inputs, test_outputs, scaler= read_data(fichero_entrenamiento)
100    if Dataset is None or train_inputs is None or train_outputs is None or test_inputs is None or test_outputs is None:
101        print("Error con el fichero")
102        return {"Error con el fichero"}
103
104    listaErrorMSE = np.zeros(shape=(5), dtype=float)
105
106    #creamos una lista con el nombre de todos los modelos
107    NombreModelo = ["SVR", "DecisionTreeRegressor", "RandomForest", "LinearRegression", "SGDRegressor"]
108
109    #entrenamos con los distintos modelo para ver cual es el mejor
110    print("Entrenando " + NombreModelo[0])
111    listaErrorMSE[0] = entrenarSVR(scaler, train_inputs, train_outputs, test_inputs, test_outputs)
112    print("Entrenando " + NombreModelo[1])
113    listaErrorMSE[1] = entrenarArbolDecision(scaler, train_inputs, train_outputs, test_inputs, test_outputs)
114    print("Entrenando " + NombreModelo[2])
115    listaErrorMSE[2] = entrenarRandomForest(scaler, train_inputs, train_outputs, test_inputs, test_outputs)
116    print("Entrenando " + NombreModelo[3])
117    listaErrorMSE[3] = entrenarRegresionLineal(scaler, train_inputs, train_outputs, test_inputs, test_outputs)
118    print("Entrenando " + NombreModelo[4])
119    listaErrorMSE[4] = entrenarSGDRegressor(scaler, train_inputs, train_outputs, test_inputs, test_outputs)

```

Imagen Anexo (Manual Código) 8: get entrenar main.py primera parte

```

121 mejorModelo = 999999
122 menorMSE = 999999
123
124 for iterador in range(len(listaErrorMSE)):#comprobamos cual genero un MSE menor
125     if listaErrorMSE[iterador] < menorMSE:
126         menorMSE = listaErrorMSE[iterador]
127         mejorModelo = iterador
128
129     print("El mejor modelo es " + NombreModelo[mejorModelo] + " con un MSE de: " + str(menorMSE))
130
131     with open(NombreModelo[iterador]+".pickle", 'rb') as fr: #cargamos el modelo ya entrenado, para guardarlo con el nombre de
132         modelo = pickle.load(fr)
133
134     #se almacena el mejor modelo
135     guardar_modelo_y_scaler(modelo, scaler, "mejorModelo")
136
137
138     return{"Modelos guardados con éxito"}

```

Imagen Anexo (Manual Código) 9: get entrenar main.py segunda parte

Tanto en la imagen 8 como 9 encontramos el get de entrenar patrones. Primeramente, se crea un fichero csv a partir de la base de datos especificada (se puede evitar que se genere este fichero con una flag, para el caso de que se encuentre creado o modificado por nosotros). Si se ha creado el fichero csv correctamente, se lee dicho fichero y se guardan los patrones en un Dataset (se almacenan en el dataset ya preprocesados).

Posteriormente, se van realizando los entrenamientos de cada modelo y almacenando el menor **MSE** que ha tenido cada uno (con su respectivo scaler).

Finalmente, se comparan todos los **MSE** para ver qué modelo ha dado un menor **MSE**. El modelo con el **MSE** menor se carga de nuevo, ya que al entrenar se guardó al ser el mejor caso. Al encontrarse el modelo cargado se almacena con el nombre de mejor modelo junto su scaler para que este pueda ser usado en la predicción.

Se devuelve un mensaje de éxito.

```

141     @app.get("/predecir/")#get que predice la iluminancia superior
142
143     def read_root(ColorSuelo: str, AlturaLuminaria: str, FlujoLuminicoTotal: str, TCC: str,
144                   IluminanciaAbajo: str, Espectro: str, ReflectanciaSuelo: str, IluminanciaSuperior: str):
145
146         patronesDict = {
147             'IluminanciaSuperior': ''
148         }
149
150         fichero_entrenamiento = "db.csv"
151         #llamamos a funcion read_data para extraer los datos del fichero csv a un formato con el que podamos trabajar
152         Dataset, train_inputs_inutilizado, train_outputs_inutilizado, test_inputs_inutilizado, test_outputs_inutilizado, scaler_inutiliza
153         if Dataset is None:
154             print("Error con el fichero")
155             return {"Error con el fichero"}
156
157
158         with open("mejorModelo.pickle", 'rb') as fr: #cargamos el modelo ya entrenado
159             model = pickle.load(fr)
160         with open("mejorModeloscaler.pickle", 'rb') as fr: #cargamos el scaler del modelo para poder preprocesar el patron
161             scaler = pickle.load(fr)

```

Imagen Anexo (Manual Código) 10: get predecir main.py primera parte

```

162
163     vector_datos_cliente = [ColorSuelo,AlturaLuminaria,FlujoLuminicoTotal,TCC,IluminanciaAbajo,Espectro,ReflectanciaSuelo,Iluminancia
164     patron_del_cliente = pd.DataFrame([vector_datos_cliente], columns = ['ColorSuelo','AlturaLuminaria','FlujoLuminicoTotal',
165                                         'TCC','IluminanciaAbajo','Espectro','ReflectanciaSuelo','IluminanciaSuperior'])
166
167     patrones_fusionados = patron_cliente mas dataset(Dataset,patron_del_cliente)
168     patrones_fusionados = patrones_fusionados[:,1:]
169     patrones_fusionados = scaler.transform(patrones_fusionados)[:, :-1]#preprocesado y escalado para ajustar el patron a predecir
170
171     #se predicen resultados, se extrane los resultados que buscamos y se desescala para que tenga sentido
172     valor_prediccion = model.predict(patrones_fusionados)#predicimos la iluminancia superior
173     valor_prediccion = np.column_stack((patrones_fusionados,valor_prediccion))
174     valor_prediccion = scaler.inverse_transform(valor_prediccion)[:, :-1]#realizamos el inverso al preprocesado y el escalado
175     valor_prediccion = np.round_(valor_prediccion, decimals=3)
176
177     patronesDict["IluminanciaSuperior"] = float(valor_prediccion)
178
179     #Devuelvo el resultado
180     return patronesDict
181
182 #####
183 #####FIN PETICIONES GET#####
184 #####

```

Imagen Anexo (Manual Código) 11: get predecir main.py segunda parte

Tanto en la imagen 10 como 11 encontramos el get de predecir patrones. Primeramente se crea un diccionario que permitirá devolver un JSON con el resultado predicho. Se lee el fichero csv sin crearlo de la base de datos (se supone que al entrenar está creado), de tal forma que tenemos un Dataset con los patrones de dicho fichero.

Se realiza el paso anterior para poder aplicar el scaler con el nuevo patrón que se ha creado. De hecho, en la línea 167, se llama a una función donde se fusiona el dataset con todos los patrones más el patrón introducido en el propio get.

Finalmente se aplica la predicción, y se desescala el patrón para devolver en un JSON el valor predicho y que se entendible, ya que si se encontrara escalado no sería entendible.

Este sería el último get, ahora se procede a explicar las funciones auxiliares.

```

189 #***** FUNCIONES DE APOYO *****
190 #***** FUNCIONES DE APOYO *****
191 #***** FUNCIONES DE APOYO *****
192
193 #funcion que extrae un fichero a partir de nuestra bd
194 def crear_fichero_entrenamiento(base_de_datos):
195     try:
196         #creamos conexion con la bd
197         conn = mysql.connector.connect(#conexion de la bd
198             user='root', password='password', host=mysql, port=3306, database=base_de_datos)
199
200         #se realiza la peticion de sql para extraer los datos de la bd y exportarlos a fichero csv
201         temp = pd.read_sql_query("Select ColorSuelo, AlturaLuminaria, FlujoLuminicoTotal, TCC, IluminanciaAbajo, Espectro,
202             temp.to_csv('db.csv', sep=',', index=False, encoding='utf-8')#como read_sql_query no funciona del todo bien con co
203
204         conn.close()#cerramos conexion con la base de datos
205         return 0
206
207     except:
208         return -1
209
210

```

Imagen Anexo (Manual Código) 12 :función crear\_fichero\_entrenamiento main.py

En la imagen 12, se puede ver la función **crear\_fichero\_entrenamiento**. En esta función se crea el fichero de entrenamiento csv que se utiliza para el entrenamiento, para ello se realiza la conexión con la base de datos, alojada en el puerto 3306. Una vez realizada la conexión, se realiza una petición a la base de datos para que nos pase todos los patrones, recibiendo únicamente los atributos especificados en la petición.

Finalmente, estos patrones recibidos se pasan a formato csv y se cierra la conexión. En caso de algún error, devuelve el valor -1.

```

213 #funcion para leer datos del fichero
214 def read_data(fichero_datos, predicción = False):
215
216     try:
217
218         Dataset = None
219         Inputs = None
220         Outputs = None
221         train_inputs = None
222         train_outputs = None
223         test_inputs = None
224         test_outputs = None
225         scaler = None
226
227         #guardamos los datos en un dataset de tipo string
228         Dataset = pd.read_csv(fichero_datos)
229         #reordenamos las columnas para poner los cardinales en las primeras filas para usar el columnTransformer
230         titulos_columnas = ["ColorSuelo", "AlturaLuminaria", "FlujoLuminicoTotal", "TCC", "IluminanciaAbajo", "Espectro", "Re
231         Dataset=Dataset.reindex(columns=titulos_columnas)
232
233         if predicción == False:
234             ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse=False), list(range(1)))], remainder= 'passth
235             Dataset = np.array(ct.fit_transform(Dataset))#transformo en numpy y necesito cambiar a pandas
236             Dataset = pd.DataFrame(Dataset,columns=["Negro", "GrisOscuro", "GrisClaro", "Blanco", "Morado", "Rojo", "VerdeClar
237                 "AlturaLuminaria", "FlujoLuminicoTotal", "TCC", "IluminanciaAbajo", "Espectro", "ReflectanciaSuelo", "Iluminan
238
239             #dividimos en 90% de datos para entreno y el resto para test
240             train, test = train_test_split(Dataset, test_size= 0.10)

```

Imagen Anexo (Manual Código) 13 :función read\_data main.py primera parte

Como se puede ver en la imagen 13, se encuentra la función **read\_data**. Esta función se encarga de leer los datos del fichero y aplicar un pequeño preprocesamiento. Primeramente, se lee el fichero almacenando los patrones en un Dataset. A dicho dataset, se le reorganizan los títulos, ya que se busca que en los primeros índices se encuentren las características categóricas para su modificación.

Una vez el dataset se encuentre ordenado. En el caso de no ser la predicción, se realiza aparte el **OneHotEncoder** que permite transformar las características categóricas como se vio explicado en puntos anteriores en este proyecto. Aparte, a las nuevas columnas generadas por el **OneHotEncoder**, se le asignan nuevas etiquetas.

Finalmente se divide el dataset en train y test, dejando un 10% del dataset original como test.

```
241     #realizamos un preprocesamiento para normalizar los datos
242     scaler = MinMaxScaler().fit(train)
243     train = scaler.transform(train)
244     test = scaler.transform(test)
245
246     #finalmente se dividen los datos en inputs y outputs
247     train_inputs=train[:, :-1]
248     train_outputs=train[:, -1:]
249     test_inputs=test[:, :-1]
250     test_outputs=test[:, -1:]
251
252     #transforma a una unica columna para evitar warnings
253     train_outputs = train_outputs.ravel()
254     test_outputs = test_outputs.ravel()
255
256     return Dataset, train_inputs, train_outputs, test_inputs, test_outputs, scaler
257
258 except:
259
260     Dataset = None
261     train_inputs = None
262     train_outputs = None
263     test_inputs = None
264     test_outputs = None
265     scaler = None
266
267     return Dataset, train_inputs, train_outputs, test_inputs, test_outputs, scaler
```

Imagen Anexo (Manual Código) 14 : función read\_data main.py segundaparte

Una vez los patrones se encuentran divididos se aplica un preprocesamiento para normalizar los datos. Para ello se crea un scaler aplicando normalización entrenado solo con los datos de entrenamiento.

Ese scaler se aplica tanto para train como test.

Tras realizar el preprocesamiento, se dividen train y test, respectivamente en inputs y outputs. Esto es necesario para poder entrenar correctamente los modelos.

Finalmente se devuelven el dataset como todos los dataset divididos.

```
271     #funcion para guardar un modelo con su scaler
272     def guardar_modelo_y_scaler(modelo, scaler, name):
273
274         #guardamos el modelo y el scaler
275         with open(name+'.pickle', 'wb') as fw:
276             pickle.dump(modelo, fw)
277         with open(name+'scaler.pickle', 'wb') as fw:
278             pickle.dump(scaler, fw)
```

Imagen Anexo (Manual Código) 15 : función guardar\_modelo\_y\_scaler main.py

Como se puede ver en la imagen 15, se encuentra la función **guardar\_modelo\_y\_scaler**. Esta función se encarga de almacenar en un fichero .pickle, el modelo y scaler recibido, poniendo a dicho scaler como nombre, el nombre pasado como parámetro.

```

281 #funcion que entrena con el modelo SVR
282 def entrenarSVR(scaler, train_inputs, train_outputs, test_inputs, test_outputs):
283
284     f=open("ResultadosParametrosSVR", "w"); ##archivo donde se almacenan los resultados de los parametros
285
286     #inicializamos parametros que usamos en el entrenamiento
287     vector = np.array([1e-2,1e-1,1e0,1e1,1e2,1e3])
288     mejor_mse = 999999
289     mae = 999999
290     mejor_C = 999999
291     mejor_Gamma = 999999
292
293     for C in vector:#entrenamos probando distintas c y gammas
294         for Gamma in vector:
295             print("C=%f y Gamma=%f" % (C, Gamma))
296
297             # Entrenamos el modelo
298             modelo = svm.SVR(kernel='rbf',C=C, gamma=Gamma)
299             modelo.fit(train_inputs, train_outputs)
300             y_pred=modelo.predict(test_inputs)
301             test_mse = mean_squared_error(test_outputs, y_pred) #MSE
302             test_mae = mean_absolute_error(test_outputs, y_pred) #MAE
303
304             f.write("MSE & MAE Final con C=%f y G=%f: \t%f %f\n" % (C, Gamma, test_mse, test_mae))#lo vamos almacenando en un fichero
305
306             if mejor_mse > test_mse:#se va guardando la mejor combinacion de parametros
307                 mejor_mse = test_mse
308                 mejor_C = C
309                 mejor_Gamma = Gamma
310                 mae = test_mae
311
312             #se almacena el modelo y el scaler
313             guardar_modelo_y_scaler(modelo,scaler,"SVR")
314
315
316     f.write("Los mejores parametros son C:"+ str(mejor_C) + " y Gamma:" + str(mejor_Gamma) + " con un MSE de " + str(mejor_mse)+" y un MAE de "+str(mae))
317     f.close()
318
319     print("*****")
320     print("Los mejores parametros son C:"+ str(mejor_C) + " y Gamma:" + str(mejor_Gamma) + " con un MSE de " + str(mejor_mse)+" y un MAE de "+str(mae))
321     print("*****")
322
323     return mejor_mse

```

Imagen Anexo (Manual Código) 16 : función entrenarSVR main.py

En la imagen 16 se puede observar la función **entrenarSVR**. Dicha función se encarga de entrenar el modelo SVR. Para ello se van recorriendo en bucle distintas combinaciones de parámetros, de tal forma que, para cada combinación se realiza el entrenamiento y predicción comparando con los datos de test, aparte se guardan en un fichero los resultados de dicha combinación.

De esta forma se obtiene el **MSE**, el cual, se compara con el mejor **MSE**, para comprobar si ha habido una mejoría. En el caso de ser cierto se almacena dicho modelo, en el caso de que no, simplemente se ignora el modelo.

Finalmente se imprimen los resultados del mejor modelo y se almacena en el fichero la mejor combinación. La función devuelve el mejor **MSE**.

```

326 #funcion que entrena con el modelo ArbolDecision
327 def entrenarArbolDecision(scaler, train_inputs, train_outputs, test_inputs, test_outputs):
328     f=open("ResultadosParametrosArbolDecision", "w"); ##archivo donde se almacenan los resultados de los parametros
329
330     #inicializamos parametros que usamos en el entrenamiento
331     v_splitter = {"best", "random"}
332     v_min_samples_split = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8}
333     v_min_samples_leaf = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8}
334     mejor_mse = 99999
335     mae = 999999
336     mejor_splitter = None
337     mejor_min_samples_split = 999999
338     mejor_min_samples_leaf = 999999
339
340
341     #entrenamos probando distintas combinaciones
342     for splitter in v_splitter:
343         for min_samples_split in v_min_samples_split:
344             for min_samples_leaf in v_min_samples_leaf:
345                 print("%s, %s, %s" % (splitter, min_samples_split, min_samples_leaf))
346
347                 #Entrenamos el modelo
348                 modelo = tree.DecisionTreeRegressor(splitter=splitter, min_samples_split=min_samples_split, min_samples_leaf=min_samples_leaf)
349                 modelo.fit(train_inputs, train_outputs)
350                 y_pred=modelo.predict(test_inputs)
351                 test_mse = mean_squared_error(test_outputs, y_pred) #MSE
352                 test_mae = mean_absolute_error(test_outputs, y_pred) #MAE
353
354                 f.write("MSE & MAE Final con splitter=%s, min_samples_split=%s y min_samples_leaf=%s: %f %f\n" % (splitter, min_samples_split, min_samples_leaf, test_mse, test_mae))
355
356                 if mejor_mse > test_mse:#se va guardando la mejor combinacion de parametros
357                     mejor_mse = test_mse
358                     mejor_splitter = splitter
359                     mejor_min_samples_split = min_samples_split
360                     mejor_min_samples_leaf = min_samples_leaf
361                     mae = test_mae
362

```

Imagen Anexo (Manual Código) 17 : función entrenarArbolDecision main.py primera parte

```

363
364         #se almacena el modelo y el scaler
365         guardar_modelo_y_scaler(modelo,scaler,"ArbolDecision")
366
367         f.write("Los mejores parametros para el arbol de decision es splitter->" + str(mejor_splitter) + ", min_samples_split=" + str(mejor_min_samples_split) + ", min_samples_leaf=" + str(mejor_min_samples_leaf) + "\n")
368         f.close()
369
370         print("*****")
371         print("Los mejores parametros para el arbol de decision es splitter->" + str(mejor_splitter) + ", min_samples_split=" + str(mejor_min_samples_split) + ", min_samples_leaf=" + str(mejor_min_samples_leaf) + "\n")
372         print("*****")
373
374     return mejor_mse

```

Imagen Anexo (Manual Código) 18 : función entrenarArbolDecision main.py segunda parte

En la imagen 17 y 18 se puede observar la función **entrenarArbolDecision**. Dicha función se encarga de entrenar el modelo **Árbol de Decisión**. Para ello se van recorriendo en bucle distintas combinaciones de parámetros, de tal forma que, para cada combinación se realiza el entrenamiento y predicción comparando con los datos de test, aparte se guardan en un fichero los resultados de dicha combinación. De esta forma se obtiene el **MSE**, el cual, se compara con el mejor **MSE**, para comprobar si ha habido una mejoría. En el caso de ser cierto se almacena dicho modelo, en el caso de que no, simplemente se ignora el modelo. Finalmente se imprimen los resultados del mejor modelo y se almacena en el fichero la mejor combinación. La función devuelve el mejor **MSE**.

```

377 #funcion que entrena con el modelo RandomForest
378 def entrenarRandomForest(scaler, train_inputs, train_outputs, test_inputs, test_outputs):
379     f=open("ResultadosParametrosRandomForest", "w"); ##archivo donde se almacenan los resultados de los parametros
380
381     #inicializamos parametros que usamos en el entrenamiento
382     v_n_estimators = {100, 300}
383     v_max_features = {0.01, 0.05, 0.1, 0.2, 0.3}
384     v_min_samples_split = {2, 50, 100}
385     v_min_samples_leaf = {2, 50, 100}
386     mejor_mse = 99999
387     mae = 999999
388     mejor_n_estimators = 999999
389     mejor_min_samples_split = 999999
390     mejor_min_samples_leaf = 999999
391     mejor_max_features = 999999
392
393     #entrenamos probando distintas combinaciones
394     for n_estimators in v_n_estimators:
395         for max_features in v_max_features:
396             for min_samples_split in v_min_samples_split:
397                 for min_samples_leaf in v_min_samples_leaf:
398                     print("n_estimators=%d, max_features=%f, min_samples_split=%f, min_samples_leaf=%f \n" % (n_estimators,
399
400                         #Entrenamos el modelo
401                         modelo = ensemble.RandomForestRegressor(min_weight_fraction_leaf=0.2, max_samples= 0.5,n_estimators=n_estimators)
402                         modelo.fit(train_inputs, train_outputs)
403                         y_pred=modelo.predict(test_inputs)
404                         test_mse = mean_squared_error(test_outputs, y_pred) #MSE
405                         test_mae = mean_absolute_error(test_outputs, y_pred) #MAE
406
407                         f.write("MSE & MAE Final con n_estimators=%d, v_max_features=%f, min_samples_split=%f, min_samples_leaf=%f \n" % (n_estimators,
408
409                         if mejor_mse > test_mse:#se va guardando la mejor combinacion de parametros
410                             mejor_mse = test_mse
411                             mae = test_mae
412                             mejor_n_estimators = n_estimators
413                             mejor_min_samples_split = min_samples_split
414                             mejor_min_samples_leaf = min_samples_leaf
415                             mejor_max_features = max_features
416
417
418         #se almacena el modelo y el scaler
419         guardar_modelo_y_scaler(modelo,scaler,"RandomForest")
420
421         f.write("Los mejores parametros para el random forest es n_estimators->" + str(mejor_n_estimators) + "max_features->" + str(mejor_max_features) + "\n")
422         f.close()
423
424         print("*****")
425         print("Los mejores parametros para el random forest es n_estimators->" + str(mejor_n_estimators) + "max_features->" + str(mejor_max_features))
426         print("*****")
427
428     return mejor_mse

```

Imagen Anexo (Manual Código) 19 : función entrenarRandomForest main.py primera parte

```

417
418         #se almacena el modelo y el scaler
419         guardar_modelo_y_scaler(modelo,scaler,"RandomForest")
420
421         f.write("Los mejores parametros para el random forest es n_estimators->" + str(mejor_n_estimators) + "max_features->" + str(mejor_max_features) + "\n")
422         f.close()
423
424         print("*****")
425         print("Los mejores parametros para el random forest es n_estimators->" + str(mejor_n_estimators) + "max_features->" + str(mejor_max_features))
426         print("*****")
427
428     return mejor_mse

```

Imagen Anexo (Manual Código) 20 : función entrenarRandomForest main.py segunda parte

En la imagen 19 y 20 se puede observar la función **entrenarRandomForest**. Dicha función se encarga de entrenar el modelo **RandomForest**. Para ello se van recorriendo en bucle distintas combinaciones de parámetros, de tal forma que, para cada combinación se realiza el entrenamiento y predicción comparando con los datos de test, aparte se guardan en un fichero los resultados de dicha combinación. De esta forma se obtiene el **MSE**, el cual, se compara con el mejor **MSE**, para comprobar si ha habido una mejoría. En el caso de ser cierto se almacena dicho modelo, en el caso de que no, simplemente se ignora el modelo. Finalmente se imprimen los resultados del mejor modelo y se almacena en el fichero la mejor combinación. La función devuelve el mejor **MSE**.

```

431 #funcion que entrena con el modelo RegresionLineal
432 def entrenarRegresionLineal(scaler, train_inputs, train_outputs, test_inputs, test_outputs):
433
434     f=open("ResultadosParametrosRegresionLineal", "w"); ##archivo donde se almacenan los resultados de los parametros
435
436     #inicializamos parametros que usamos en el entrenamiento
437     mejor_mse = 999999
438     mae = 999999
439
440     #Entrenamos el modelo
441     modelo = linear_model.LinearRegression()
442     modelo.fit(train_inputs, train_outputs)
443     y_pred=modelo.predict(test_inputs)
444     test_mse = mean_squared_error(test_outputs, y_pred) #MSE
445     test_mae = mean_absolute_error(test_outputs, y_pred) #MAE
446
447     mejor_mse = test_mse
448     mae = test_mae
449
450     f.write("MSE & MAE Final con para la regresion lineal: \t%f %f\n" % (test_mse, test_mae))
451     f.close()
452
453     #se almacena el modelo y el scaler
454     guardar_modelo_y_scaler(modelo,scaler,"RegresionLineal")
455
456     print("*****")
457     print("La regresion lineal no tiene parametros que buscar, siendo su MSE: "+ str(mejor_mse)+" y un MAE de "+str(mae))
458     print("*****")
459
460     return mejor_mse
461

```

Imagen Anexo (Manual Código) 21 : función entrenarRegresionLineal main.py

En la imagen 21 se puede observar la función **entrenarRegresionLineal**. Dicha función se encarga de entrenar el modelo **RegresionLineal**. Esta función es distinta a las demás ya que no dispone de combinaciones de parámetros. Por lo tanto solo se obtiene el MSE.

Finalmente se imprimen los resultados del modelo y se almacena en el fichero la mejor combinación. También se almacena dicho modelo con su respectivo scaler. La función devuelve el mejor MSE.

```

463 #funcion que entrena con el modelo SGDRegressor
464 def entrenarSDGRegressor(scaler, train_inputs, train_outputs, test_inputs, test_outputs):
465
466     f=open("ResultadosParametrosSGDRegressor", "w"); ##archivo donde se almacenan los resultados de los parametros
467
468     #inicializamos parametros que usamos en el entrenamiento
469     v_validation = {0.1, 0.2, 0.3, 0.4}
470     v_early_stopping = {True, False}
471     v_shuffle = {True, False}
472     mejor_mse = 999999
473     mae = 999999
474     mejor_validation = 999999
475     mejor_early_stopping = None
476     mejor_shuffle = None
477     mejor_alpha = 999999
478
479     #entrenamos probando distintas combinaciones
480     for early_stopping in v_early_stopping:
481         for validation in v_validation:
482             for shuffle in v_shuffle:
483                 for alpha in range(1,21,1):
484                     print("early_stopping=%r, validation=%f, shuffle=%r, alpha=%f \n" % (early_stopping, validation, shuffle, alpha))
485
486                     #Entrenamos el modelo
487                     modelo = linear_model.SGDRegressor(early_stopping=early_stopping, validation_fraction=validation, shuffle=shuffle)
488                     modelo.fit(train_inputs, train_outputs)
489                     y_pred=modelo.predict(test_inputs)
490                     test_mse = mean_squared_error(test_outputs, y_pred)
491                     test_mae = mean_absolute_error(test_outputs, y_pred) #MAE
492
493                     f.write("MSE & MAE Final con early_stopping=%r, validation=%f, shuffle=%r, alpha=%f= \t%f %f\n \n" % (early_stopping, validation, shuffle, alpha, test_mse, test_mae))
494
495                     if mejor_mse > test_mse:#se va guardando la mejor combinacion de parametros
496                         mejor_mse = test_mse
497                         mae = test_mae
498                         mejor_early_stopping = early_stopping
499                         mejor_validation = validation
500                         mejor_shuffle = shuffle
501                         mejor_alpha = alpha/10000

```

Imagen Anexo (Manual Código) 22 : función entrenarSDGRegressor main.py primera parte

```

502
503     #se almacena el modelo y el scaler
504     guardar_modelo_y_scaler(modelo,scaler,"SGDRegressor")
505
506
507     f.write("Los mejores parametros para el SGDRegressor es early_stopping->" + str(mejor_early_stopping) + " , validation->" + str(validation))
508     f.close()
509
510     print("*****")
511     print("Los mejores parametros para el SGDRegressor es early_stopping->" + str(mejor_early_stopping) + " , validation->" + str(validation))
512     print("*****")
513
514     return mejor_mse

```

Imagen Anexo (Manual Código) 23 : función entrenarSGDRegressor main.py segunda parte

En la imagen 22 y 23 se puede observar la función **entrenarSGDRegressor**. Dicha función se encarga de entrenar el modelo **SGDRegressor**. Para ello se van recorriendo en bucle distintas combinaciones de parámetros, de tal forma que, para cada combinación se realiza el entrenamiento y predicción comparando con los datos de test, aparte se guardan en un fichero los resultados de dicha combinación. De esta forma se obtiene el **MSE**, el cual, se compara con el mejor **MSE**, para comprobar si ha habido una mejoría. En el caso de ser cierto se almacena dicho modelo, en el caso de que no, simplemente se ignora el modelo. Finalmente se imprimen los resultados del mejor modelo y se almacena en el fichero la mejor combinación. La función devuelve el mejor **MSE**.

```

517 #funcion para juntar el patron del cliente con el dataset
518 def patron_cliente_mas_dataset(Dataset,patron_del_cliente):
519
520     Dataset = pd.concat((Dataset, patron_del_cliente))
521
522     #aplicamos el onehotencoder
523     ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse=False), list(range(1)))], remainder= 'passthrough')
524     Dataset = np.array(ct.fit_transform(Dataset))#transformo en numpy y necesito cambiar a pandas
525     Dataset = pd.DataFrame(Dataset,columns=["Negro", "GrisOscuro", "GrisClaro", "Blanco", "Morado", "Rojo", "VerdeClaro", "Verde", "AlturaLuminaria", "FlujoLuminicoTotal", "TCC", "IluminanciaAbajo", "Espectro", "ReflectanciaSuelo", "IluminanciaSuperior"])
526
527
528     return Dataset
529
530
531
532
533 #####*****#####*****#####*****#####*****#####*****#####*****#####*****#####
534 #####*****#####*****#####*****#####*****#####*****#####*****#####*****#####
535 #####*****#####*****#####*****#####*****#####*****#####*****#####*****#####

```

Imagen Anexo (Manual Código) 24: función patron\_cliente\_mas\_dataset main.py

En la imagen 24 se puede observar la función **patron\_cliente\_mas\_dataset**. Dicha función se encarga de fusionar el patrón recibido por el cliente con el Dataset de patrones para poder aplicar el scaler.

Simplemente se concatenan los dos dataset y se aplica el **OneHotEncoder** para que tenga el mismo preprocessamiento que en la función **read\_data**.

Se devuelve el dataset fusionado y preprocessado.

Con esta última función quedan brevemente explicadas todas las peticiones “get” y todas las funciones auxiliares.

## 4. preprocesamiento.py

Este fichero es un fichero auxiliar que se encuentra fuera de los servicios del docker compose. Su principal función es realizar el preprocesamiento de los datos obtenidos en la adquisición. Con las siguientes imágenes se explica su funcionamiento.

```
1 #paquetes necesarios para el funcionamiento del programa
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt#paquete para mostrar histogramas
5 import seaborn as sns#paquete para mostrar el mapa de correlaciones
6 from sklearn import preprocessing#paquete de scikit con el estandarizado, normalizacion y el onehotencoder
7 from sklearn.preprocessing import MinMaxScaler, StandardScaler
8 from sklearn.compose import ColumnTransformer
9 from sklearn.preprocessing import OneHotEncoder
10
11 import mysql.connector #paquete para insertar en la bd de mysql
12
```

Imagen Anexo (Manual Código) 25: import preprocesamiento.py

En la imagen 25 se pueden ver los import necesarios para el correcto funcionamiento del código creado en preprocesamiento.py

```
14 #funcion main para realizar el preprocesamiento necesario de los datos
15 def main():
16
17     crear_fichero()#creamos un fichero extraido de la bd para trabajar
18     fichero_datos = "db.csv"
19
20     Dataset = read_data(fichero_datos)#se leen los datos del fichero creado anteriormente para insertarlos
21     if Dataset is None:
22         print("Error al hacer el dataset a partir del fichero creado de la bd")
23         return
24
25     Dataset.hist(figsize=(14,14), xrot=45)#muestro histograma del dataset
26     plt.show()
27
28     #guardamos las correlaciones y las mostramos en un mapa
29     Datasetcorr = Dataset.corr()
30     plt.figure(figsize=(14,14))
31     sns.heatmap(Datasetcorr, cmap='RdBu_r')
32     plt.show()
33
34     #realizamos normalizacion y estandarización del dataset, es necesario que al estandarizar y normalizar
35     #los datos se quiten la ultima columna que es la id
36     x = Dataset.values
37     DatasetNormalizado=x[:, :-1]
38     DatasetEstandarizado=x[:, :-1]
39     auxiliarEstandar = pd.DataFrame(DatasetEstandarizado)
40
41     minmax_scaler = MinMaxScaler()
42     x_scaled = minmax_scaler.fit_transform(DatasetNormalizado)
43     DatasetNormalizado = pd.DataFrame(x_scaled)
44     DatasetNormalizado.hist(figsize=(14,14), xrot=45)
45     plt.show()
46
47     estandarizado_scaled= StandardScaler()
48     estandarizadoFinal= estandarizado_scaled.fit_transform(DatasetEstandarizado)
49     nombres=auxiliarEstandar.columns
50     EstandarizadoFinal =pd.DataFrame(estandarizadoFinal, columns=nombres)
51     EstandarizadoFinal.hist(figsize=(14,14), xrot=45)
52     plt.show()
```

Imagen Anexo (Manual Código) 26: funcion main preprocesamiento.py

En la imagen 26 se puede ver la función **main**. Dicha función es la función principal de este programa.

Primeramente se crea el fichero csv. Posteriormente se leen los datos de dicho fichero para que se encuentren almacenados en un Dataset.

Una vez los datos se encuentran almacenados en un Dataset, se crea el primer histograma y se muestra (con **show**), después se crea el mapa de correlaciones y se muestra (en la línea 31 se crea el mapa de correlaciones).

Finalmente se aplica normalización con **MinMaxScaler()** o estandarización con **StandardScaler()** y se crean y muestran sus histogramas.

```
54 #funcion que extrae un fichero a partir de nuestra bd
55 def crear_fichero():
56     #Creamos conexion con la bd
57     conn = mysql.connector.connect(#conexion de la bd
58         user='root', password='password', host="127.0.0.1", port=3306, database='db')
59
60     #se realiza la peticion de sql para extraer los datos de la bd y exportarlos a fichero csv
61     temp = pd.read_sql_query("Select TipoLuminaria, ColorSuelo, AlturaLuminaria, FlujoLuminicoTotal, TCC, IluminanciaAbajo, Espectro, ReflectanciaSuelo from db")
62     temp.to_csv('db.csv', sep=',', index=False, encoding='utf-8')#como read_sql_query no funciona
63
64     conn.close()#cerramos conexion con la base de datos
```

Imagen Anexo (Manual Código) 27: función crear\_fichero preprocesamiento.py

La imagen 27 coincide con la función **crear\_fichero**. Esta función a su vez coincide con la función ya explicada **crear\_fichero\_entrenamiento** del programa **main.py**

```
67 #funcion para leer datos del fichero
68 def read_data(fichero_datos):
69
70     Dataset = None
71
72     #guardamos los datos en un dataset de tipo string
73     Dataset = pd.read_csv(fichero_datos)
74     #reordeno las columnas para poner los cardinales en las primeras filas para usar el columnTransformer
75     titulos_columnas = ["TipoLuminaria", "ColorSuelo", "AlturaLuminaria", "FlujoLuminicoTotal", "TCC", "IluminanciaAbajo", "Espectro", "ReflectanciaSuelo"]
76     Dataset=Dataset.reindex(columns=titulos_columnas)
77
78     ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse=False), list(range(2)))], remainder='passthrough')
79     Dataset = np.array(ct.fit_transform(Dataset))#transformo en numpy y necesito cambiar a pandas
80     Dataset = pd.DataFrame(Dataset,columns=["Tipo1", "Tipo2", "Tipo3", "Tipo4", "Tipo5", "Tipo6", "Tipo7", "Negro", "GrisOscur", "GrisClaro", "Blanco", "Morado", "Rojo", "VerdeClaro", "Verde", "VerdeOscuro", "AlturaLuminaria", "FlujoLuminicoTotal", "TCC", "IluminanciaAbajo", "Espectro", "ReflectanciaSuelo"])
81
82     return Dataset
83
84
85
86
87 if __name__ == "__main__":
88     main()
```

Imagen Anexo (Manual Código) 28: función read\_data preprocesamiento.py

La imagen 28 coincide con la función **read\_data**. Esta función a su vez coincide con la función ya explicada **read\_data** del programa **main.py**

## 5. generador\_grafica.py

Este fichero es un fichero auxiliar que se encuentra fuera de los servicios del docker compose. Su principal función es generar dos gráficas comparativas entre datos reales y datos predichos por el modelo. Con las siguientes imágenes se explica su funcionamiento.

```
1  #paquete utilizados
2  import requests
3  import json
4  import numpy as np
5  import pandas as pd
6  from matplotlib import pyplot as plt
```

Imagen Anexo (Manual Código) 29: import generador\_grafica.py

En la imagen 29 se pueden ver los import necesarios para el correcto funcionamiento del código creado en preprocesamiento.py

```
8  #funcion main para generar las graficas
9  def main():
10
11     Dataset = read_data("test_tmp.csv")#se leen los datos del fichero creado anteriormente para insertarlo en un Dataset
12     if Dataset is None:
13         print("Error al hacer el dataset a partir del fichero")
14         return
15
16     IluminanciaSuperiorEsperada = []
17     IluminanciaSuperiorObtenida = []
18     FlujoSuperiorEsperado = []
19     FlujoSuperiorObtenido = []
20     FlujoLuminicoTotalEsperado = []
21     IluminanciaAbajoEsperada = []
22     FHSIEsperado = []
23     FHSIObtido = []
24
25     #recorremos el Dataset leido para pasale los datos a la api
26     for i in range(0, len(Dataset), 1):
27
28         #parametros para para la peticion get
29         getParameters_prediccion = {
30             "base_de_datos": "db",
31             "ColorSuelo": Dataset.loc[i]["ColorSuelo"],
32             "AlturaLuminaria": Dataset.loc[i]["AlturaLuminaria"],
33             "FlujoLuminicoTotal": Dataset.loc[i]["FlujoLuminicoTotal"],
34             "TCC": Dataset.loc[i]["TCC"],
35             "IluminanciaAbajo": Dataset.loc[i]["IluminanciaAbajo"],
36             "Espectro": Dataset.loc[i]["Espectro"],
37             "ReflectanciaSuelo": Dataset.loc[i]["ReflectanciaSuelo"],
38             "IluminanciaSuperior": 0,
39         }
```

Imagen Anexo (Manual Código) 30: funcion main generador\_grafica.py primera parte

```

41     #get para pedir el patron a predecir para cada uno
42     patron_con_prediccion = requests.get("http://127.0.0.1:8000/predecir", #peticion al modulo para obtener la prediccion
43         params = getParameters_prediccion
44     )
45     if not patron_con_prediccion.status_code == 200:#se comprueba si ha fallado
46         raise Exception('Incorrect reply from Ree API. Status code: {}. Text: {}'.format(patron_con_prediccion.status_code, patron_co
47             return {"Error al extraer datos para la prediccion"}
48     patron_solucion = (patron_con_prediccion.json())
49     patron_solucion = patron_solucion['IluminanciaSuperior']
50
51     #almacenamos datos necesarios para poder calcular despues el flujo superior y FHSI
52     IluminanciaAbajoEsperada.append(DataSet.loc[i]["IluminanciaAbajo"])
53     FlujoLuminicoTotalEsperado.append(DataSet.loc[i]["FlujoLuminicoTotal"])
54
55     IluminanciaSuperiorEsperada.append(DataSet.loc[i]["IluminanciaSuperior"])
56     IluminanciaSuperiorObtenida.append(patron_solucion)
57
58
59
60     for w in range(len(IluminanciaAbajoEsperada)):
61         #realizamos los calculos de FHSI y FlujoSuperior
62         FlujoSuperiorEsperado.append(FlujoLuminicoTotalEsperado[w] * (IluminanciaSuperiorEsperada[w])/IluminanciaAbajoEsperada[w])
63         FlujoSuperiorObtenido.append(FlujoLuminicoTotalEsperado[w] * (IluminanciaSuperiorObtenida[w])/IluminanciaAbajoEsperada[w])
64
65         FHSIEsperado.append(round(((FlujoSuperiorEsperado[w]/FlujoLuminicoTotalEsperado[w]) * 100), 3))
66         FHSIObtendido.append(round(((FlujoSuperiorObtenido[w]/FlujoLuminicoTotalEsperado[w]) * 100), 3))

```

Imagen Anexo (Manual Código) 31: funcion main generador\_grafica.py segunda parte

```

68     #grafica para la iluminancia
69     fig, ax = plt.subplots(figsize=(20, 10))
70     ax.autoscale(enable=None, axis="x", tight=True)
71     ax.set_ylimits(bottom=0, top=300)
72     ax.plot( IluminanciaSuperiorObtenida, 'r-', label='predicho')
73     ax.plot( IluminanciaSuperiorEsperada, 'b-', label='real')
74     plt.ylabel('IluminanciaSuperior')
75     plt.xlabel('patron')
76     plt.xticks(rotation = '90');
77     plt.legend()
78     plt.savefig("testIluminacion.png",bbox_inches='tight', dpi=100)
79
80     #grafica para la contaminacion luminica
81     fig, ax = plt.subplots(figsize=(20, 10))
82     ax.autoscale(enable=None, axis="x", tight=True)
83     ax.set_ylimits(bottom=0, top=30)
84     ax.plot( FHSIObtendido, 'r-', label='predicho')
85     ax.plot( FHSIEsperado, 'b-', label='real')
86     plt.ylabel('ContaminacionLuminica')
87     plt.xlabel('patron')
88     plt.xticks(rotation = '90');
89     plt.legend()
90     plt.savefig("testContaminacionLuminica.png",bbox_inches='tight', dpi=100)
91

```

Imagen Anexo (Manual Código) 32: funcion main generador\_grafica.py tercera parte

En la imagen 30, 31 y 32 se puede observar la función **main**. Dicha función es la función principal de este programa. Primeramente se lee el fichero csv. Este fichero contiene los patrones que se desean predecir y que contienen su valor original. Estos patrones han sido escogidos del fichero original de tal forma que han sido eliminados del entrenamiento, ya que si se entrena con ellos el modelo ya los reconocería.

Una vez tenemos el Dataset con los valores a predecir, se van realizando peticiones “get” al modelo para que prediga cada uno de los patrones. Dicho resultado se va almacenando en un vector del tamaño de todos los patrones enviados.

Una vez tenemos las iluminancias predecidas, se calcula el índice de contaminación lumínica esperado y real. De esta forma ya tendríamos todos los datos necesarios para realizar las gráficas.

Finalmente, se crean **dos gráficas** donde en la primera se compara la iluminancia predicha frente a la real, y en la segunda, se compara el porcentaje de contaminación lumínica predicha frente a la real.

```
94 #funcion para leer datos del fichero
95 def read_data(fichero_datos):
96     Dataset = None
97
98     #guardamos los datos en un dataset de tipo string
99     Dataset = pd.read_csv(fichero_datos)
100    #reordeno las columnas para poner los cardinales en las primeras filas para usar el columnTransformer
101    titulos_columnas = ["ColorSuelo", "AlturaLuminaria", "FlujoLuminicoTotal", "TCC", "IluminanciaAbajo", "Espectro", "Re
102    Dataset=Dataset.reindex(columns=titulos_columnas)
103
104    return Dataset
105
106
107
108 if __name__ == "__main__":
109     main()
```

Imagen Anexo (Manual Código) 33: función read\_data generador\_grafica.py

La imagen 33 coincide con la función **read\_data**. Esta función a su vez coincide con la función ya explicada **read\_data** del programa **main.py**

## 6. ejemplocliente.py

Este fichero es un fichero auxiliar que se encuentra fuera de los servicios del docker compose. Su principal función es simular un cliente de forma sencilla, es decir, un programa donde un cliente quiere predecir una combinación de parámetros, en vez de realizar una petición “get” de predicción directamente desde el navegador.

```
1 import requests
2 import json
3 import numpy as np
4 import os
```

Imagen Anexo (Manual Código) 34: import ejemplocliente.py

En la imagen 34 se pueden ver los import necesarios para el correcto funcionamiento del código creado en ejemplocliente.py

```
7 #funcion main para realizar el preprocesamiento necesario de los datos
8 def main():
9     flag = True
10    while (flag):
11        print("Introduzca ColorSuelo: ")
12        print("1.Negro\n2.GrisOscuro\n3.GrisClaro\n4.Blanco\n5.Morado\n6.Rojo\n7.Verde")
13        IndiceColorSuelo = input()
14        os.system ("clear")
15        print(IndiceColorSuelo)
16        try:
17            print("hola")
18            IndiceColorSuelo = int(IndiceColorSuelo)
19            print(IndiceColorSuelo)
20            if (IndiceColorSuelo > 0 and IndiceColorSuelo < 12):
21                flag = False
22                print("holaz")
23                if IndiceColorSuelo == 1:
24                    ColorSuelo = "negro"
25                elif IndiceColorSuelo == 2:
26                    ColorSuelo = "gris oscuro"
27                elif IndiceColorSuelo == 3:
28                    ColorSuelo = "gris claro"
29                elif IndiceColorSuelo == 4:
30                    ColorSuelo = "blanco"
31                elif IndiceColorSuelo == 5:
32                    ColorSuelo = "morado"
33                elif IndiceColorSuelo == 6:
34                    ColorSuelo = "rojo"
35                elif IndiceColorSuelo == 7:
36                    ColorSuelo = "verde claro"
37                elif IndiceColorSuelo == 8:
38                    ColorSuelo = "verde"
39                elif IndiceColorSuelo == 9:
40                    ColorSuelo = "Verde oscuro"
41                elif IndiceColorSuelo == 10:
42                    ColorSuelo = "marrón claro"
43                elif IndiceColorSuelo == 11:
44                    ColorSuelo = "marrón oscuro"
45                else:
46                    print("Valor introducido erroneo.")
47
48
```

Imagen Anexo (Manual Código) 35: función main ejemplocliente.py primera parte

```
49         except:  
50             print("Valor introducido erroneo.")  
51             os.system ("clear")  
52             AlturaLuminaria = input("Introduzca AlturaLuminaria: ")  
53             os.system ("clear")  
54             FlujoLuminicoTotal = input("Introduzca FlujoLuminicoTotal: ")  
55             os.system ("clear")  
56             TCC = input("Introduzca TCC: ")  
57             os.system ("clear")  
58             IluminanciaAbajo = input("Introduzca IluminanciaAbajo: ")  
59             os.system ("clear")  
60             Espectro = input("Introduzca Espectro: ")  
61             os.system ("clear")  
62             ReflectanciaSuelo = input("Introduzca ReflectanciaSuelo: ")  
63             os.system ("clear")  
64  
65             """#Philips Coreline Malaga LED BRP102 LED55/740 II DM,negro,160.0,4600.0  
66             ColorSuelo = "negro"  
67             AlturaLuminaria = 160.0  
68             FlujoLuminicoTotal = 4600  
69             TCC = 4033.0  
70             IluminanciaAbajo = 773.0  
71             Espectro = 452.0  
72             ReflectanciaSuelo = 4.0"""  
73  
74             #parametros para la petición get  
75             getParameters_prediccion = {  
76                 "base_de_datos": "db",  
77                 "ColorSuelo": ColorSuelo,  
78                 "AlturaLuminaria": AlturaLuminaria,  
79                 "FlujoLuminicoTotal": FlujoLuminicoTotal,  
80                 "TCC": TCC,  
81                 "IluminanciaAbajo": IluminanciaAbajo,  
82                 "Espectro": Espectro,  
83                 "ReflectanciaSuelo": ReflectanciaSuelo,  
84                 "IluminanciaSuperior": 0,  
85             }
```

Imagen Anexo (Manual Código) 36: función main ejemplociente.py segunda parte

```
88     #get para pedir el patron a predecir
89     patron_con_prediccion = requests.get("http://127.0.0.1:8000/predecir",#peticion al mod
90         params = getParameters_prediccion
91     )
92     if not patron_con_prediccion.status_code == 200:#se comprueba si ha fallado
93         raise Exception("Incorrect reply from Ree API. Status code: {}. Text: {}".format(p
94             return {"Error al extraer datos para la prediccion"}
95     patron_solucion = (patron_con_prediccion.json())
96     IluminanciaSuperior = patron_solucion['IluminanciaSuperior']
97
98     #mostramos el FlujoSuperior y el FHSI una vez calculado
99     print("La IluminanciaSuperior es : ", "{0:.3f}".format(IluminanciaSuperior))
100    FlujoSuperior = float(FlujoLuminicoTotal) * (IluminanciaSuperior/float(IluminanciaAbaj
101    print("La FlujoSuperior es : ", "{0:.3f}".format(FlujoSuperior))
102    FHSI = (FlujoSuperior/float(FlujoLuminicoTotal)) * 100
103    print("La contaminación Luminica es : ", "{0:.3f}".format(FHSI), "%")
104
105    #dependiendo del flujo mostramos diferentes mensajes
106    if FHSI > 25:
107        print("Como la contaminacion luminica es superior al 25%, dicha luminaria no podri
108    elif FHSI > 15:
109        print("Como la contaminacion luminica es superior al 15%, dicha luminaria solo pod
110    elif FHSI > 5:
111        print("Como la contaminacion luminica es superior al 5%, dicha luminaria solo podr
112    elif FHSI >1:
113        print("Como la contaminacion luminica es superior al 1%, dicha luminaria solo podr
114    else:
115        print("Como la contaminacion luminica es inferior al 1%, dicha luminaria podria us
116
117
118    if __name__ == "__main__":
119        main()
```

Imagen Anexo (Manual Código) 37: función main ejemplocliente.py tercera parte

En la imagen 35, 36 y 37 se puede observar la función **main**. Dicha función es la función principal de este programa. Primeramente se crea un menú simple donde se van pidiendo los parámetros al usuario. Una vez se obtienen todos los parámetros, se realiza la petición “*get*” de predicción al servicio con los modelos.

La respuesta obtenida de la petición “*get*”, se procesa para extraer el flujo superior y la contaminación lumínica, ya que la respuesta es la iluminancia superior. Por ello se aplican las fórmulas explicadas durante la documentación.

Finalmente, para mayor facilidad para el usuario, se le comenta para qué tipos de E estaría permitida dicha combinación de características para dicha luminaria.

