Universidad Internacional de La Rioja (UNIR)

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Análisis y Visualización de Datos Masivos

Sistema de predicción de los niveles de contaminación en Madrid

Trabajo Fin de Máster

presentado por: Martínez-Toledano Torres, Luz María

Director: Villaplana Pérez, Miguel

Ciudad: Madrid

Fecha: 25 de julio de 2019

Gracias a todos aquellos que en algún momento han creído en mí y me han apoyado siempre, con una sonrisa.

"La contaminación nunca debería ser el precio de la prosperidad"

- Al Gore

Abstract

The objective of this work is to build predictive models that help to estimate the level of air pollution in Madrid, Spain as far in advance as possible. The models are generated from meteorological variables and the values of the magnitudes obtained from the air quality stations. To construct the models, decision trees and neural networks are used. A dashboard is designed with the objective of managing measurements and prediction values in a simple way. After obtaining the results, it can be observed that they are affected by the imbalance in the data set. Therefore, other options should be explored such as the use of Python specific libraries for unbalanced data and the inclusion of other predictor variables, in addition to developing software for model feedback.

Keywords: predictive model, artificial intelligence, dashboard, pollution, air quality

Resumen

El objetivo de este trabajo es construir modelos predictivos que ayuden a estimar con la mayor anterioridad posible el nivel de contaminación del aire de Madrid, España. Los modelos se generan a partir de variables meteorológicas y los valores de las magnitudes obtenidos de las estaciones de calidad del aire. Para construir los modelos se usan árboles de descisión y redes neuronales. Se diseña un *dashboard* con el objetivo de gestionar de manera sencilla las medidas y valores de predicción. Tras obtener los resultados, se puede observar que se ven perjudicados por el desequilibrio en el conjunto de datos. Por tanto, se deben explorar otras opciones como el uso de librerías específicas de Python para datos desbalanceados y la inclusión de otras variables predictoras además de desarrollar el software para la retroalimentación de los modelos.

Palabras Clave: modelo predictivo, inteligencia artificial, dashboard, contaminación, calidad del aire

Índice de ilustraciones

Índice de tablas

Lista de acrónimos

ACVA	Accidentes cerebrovasculares
AEMET	Agencia Estatal de Meteorología
API KEY	Application Programming Interface. KEY
API REST	Application Programming Interface. REpresentational State Transfer
CO_2	Dióxido de carbono
CO	Óxido de carbono
CSV	Comma Separated Values
DCAT	Data Catalogue Vocabulary
DOT	Archivos de documento de plantilla de Word
GEO	Geography File
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
NO	Óxido de nitrógeno
NO_2	Dióxido de nitrógeno
O_3	Óxido troposférico
PM	Partículas en suspensión
SO_2	Dióxido de azufre
SO_3	Trióxido de azufre
TXT	Textfile
UCLM	Universidad de Castilla La Mancha
URL	Uniform Resource Locator (Localizador Uniforme de Recursos)
XLS	Microsoft Excel File
XML	eXtensible Markup Language

Índice de contenido

1. Introducción

La contaminación atmosférica, actualmente, es uno de los mayores problemas ya que no sólo afecta a nuestro entorno, si no que se refleja en el deterioro de la salud de los seres vivos. En este trabajo se aborda la construcción de un modelo de predicción de la contaminación atmosférica en Madrid junto con un *dashboard* que nos permita gestionar los resultados. Para ello, contaremos con datos procedentes de las mediciones recogidas por las estaciones de control de la calidad del aire de la ciudad y el clima. Incialmente se capturarán los datos. Después, se estudiarán y procesarán para poder aplicarles técnicas de inteligencia artificial, como árboles de decisión y redes neuronales. A raíz de los resultados se obtendrán las conclusiones. Finalmente se propondrán las líneas de futuro con las que se conseguirán mejores resultados en las predicciones.

1.1. Motivación

El cambio climático debería ser una de las mayores preocupaciones a tener en cuenta hoy en día, pero es un concepto diferente a la contaminación del aire, que a veces se confunden entre si al estar muy unidos. El cambio climático, ocurre a nivel global, es la transformación del clima del planeta Tierra, por diferentes causas. Estas pueden ser naturales o generadas por el ser humano. Sus consecuencias son tales como la subida de la temperatura, el deshielo de los polos, el aumento del nivel del mar, fenómenos meteorológicos más extremos, la desertización de los suelos, etc. Esta transformación desencadena no sólo problemas ambientales, sino que tiene consecuencias negativas para la sociedad en sí. Respecto a la contaminación del aire, que impacta más localmente al surgir normalmente en núcleos urbanos, existe porque hay partículas nocivas para el ser humano en él, además de la vegetación y el resto de animales. El mayor contribuyente es la contaminación atmosférica con los óxidos de nitrógeno, gases de ozono troposférico, óxidos de azufre, partículas en suspensión, etc.

Ambos problemas derivan en gran parte de acciones llevadas a cabo por el hombre. Altas emisiones provocadas por el empleo de petróleo, gas y carbón para los transportes y la industria, quemas de bosques y talas incontroladas de éstos, aerosoles, radiación, abuso de una alimentación carnívora principalmente basada en la ternera, etc. Pero sin duda, el mayor causante es el actual modelo energético y el exceso de combustibles fósiles. El aumento de emisiones de CO_2 produce el calentamiento global que a su vez provoca el cambio climático,

mientras que la producción de NO, NO_2, SO_2, SO_3 o PM son los culpables de que haya contaminación en el aire.

Como ya hemos comentado, ambos tienen un impacto altamente negativo provocando inundaciones y sequías que a su vez hacen que existan muchos problemas en la agricultura, deforestación y extinción de especies. Nace la hambruna y con ello los enfrentamientos en la sociedad e incluso guerras.

La contaminación del aire afecta gravemente a la salud humana causando muchas muertes al año en el mundo. Es responsable de parte de los cánceres de pulmón, ataques al corazón e infartos cerebrales. Todo ello conlleva un mayor gasto sanitario y reducciones en el tiempo trabajado, traduciéndose todo ello en pérdidas de dinero. El problema va en aumento y cada vez es peor. Al margen de las opiniones de la sociedad que piensan que el daño ya está hecho, es irreversible y sólo podemos frenarlo, existen otras más optimistas, que abogan por que si llevamos a cabo las medidas correctas, a muy largo plazo, esta situación pueda llegar a ser reversible. En cualquier caso, necesitamos encontrar y aplicar soluciones con urgencia.

En el futuro se tendría que contar únicamente con modelos energéticos sostenibles y más eficientes como las energías renovables. Esta solución es viable, pero a largo plazo. Mientras que se consigue este objetivo tan ambicioso, pero real y alcanzable, se pueden aplicar otras medidas con resultados a corto plazo, como: restricciones al tráfico, promover el uso del transporte público (dando ejemplo y siendo éstos medios eléctricos o por lo menos híbridos), fomentar el uso de la bicicleta habilitando carriles bici en las ciudades, vigilar de cerca la contaminación, etc.

Conscientes de este problema, se quiere aportar un granito de arena con este trabajo a las medidas que ya aplica el Ayuntamiento de Madrid para combatir la contaminación en el aire. Se llevará a cabo un estudio que intentará predecir el nivel de contaminación en el aire de Madrid con la mayor antelación que sea posible. Además, se creará un *dashboard* donde se visualizará tanto la contaminación en tiempo real como los niveles de predicción.

1.2. Planteamiento del trabajo

Muy a nuestro pesar, por el momento no se puede erradicar la contaminación pero sí controlar y aplicar las medidas oportunas, por severas que parezcan, para que ésta no pase de cierto límite. Una solución es predecir con el mayor margen posible los niveles que va a

alcanzar la contaminación y con ello poder avisar a la población de las acciones que se van a llevar a cabo, como por ejemplo las restricciones de tráfico. De esta forma, la ciudadanía puede tener una mejor planificación de su vida en lo que, por ejemplo, al transporte se refiere. La predicción se hará con inteligencia artificial, más concretamente con árboles de decisión y redes neuronales.

1.3. Estructura del trabajo

En el cuadro 1 se resume la estructura del contenido de la memoria. El contexto y el estado del arte se explican en el capítulo 2, mientras que los objetivos se detallan en el 3. La metodología de trabajo la encontraremos en el capítulo 4 y todo el desarrollo específico de la contribución en el 5. La evaluación junto con las conclusiones y líneas de futuro se explican en los capítulos 6 y 7, respectivamente. Finalmente en el capítulo 8 se hallan las referencias bibliográficas y en el 9 los anexos del trabajo, en los que se incluye todo el código desarrollado que conforma el *software*.

1	Introducción
2	Contexto y estado del arte
3	Objetivos
4	Metodología del trabajo
5	Desarrollo específico de la contribución
6	Evaluación
7	Conclusiones y líneas de futuro
8	Bibliografía
9	Anexos

Cuadro 1: Estructura del contenido de la memoria

2. Contexto y estado del arte

Como una causa de muerte actual tanto en países desarollados como subdesarrollados, la contaminación del aire es motivo de estudio y análisis desde hace años.

2.1. Contexto

La contaminación y sus consecuencias nos atañen a todos. Aparte de los cambios negativos medio ambientales que se producen, también perjudican nuestra salud.

En el aire existen partículas en suspensión (PM). La combustión de carburantes fósiles generada por el tráfico, en especial los vehículos, pueden producir diversos tipos de partículas, partículas más grandes por la liberación de materiales mal quemados (cenizas volátiles) o partículas más finas formadas por la condensación de materiales vaporizados durante la combustión. Pueden existir otras partículas secundarias procedentes de reacciones atmosféricas que afectan a contaminantes desprendidos como gases. Se dividen en dos grupos atendiendo a su tamaño: las PM_{10}, menos perjudiciales para la salud ya que normalemnte sólo penetran hasta las vías respiratorias bajas y tienen un tamaño menor de $10\mu g$; y las $PM_{2,5}$ que son partículas ”respirables” menores de $2,5\mu g$ que pueden penetrar hasta la zona de intercambio de gases del pulmón. Las PM_{10} son partículas sólidas o líquidas de polvo, cenizas, hollín, metálicas, cemento, tierra o polen dispersas en la atmósfera. Están compuestas principalmente por compuestos inorgánicos como metales pesados y material orgánico asociado a partículas de carbono, como el hollín. En cambio, las $PM_{2,5}$ proceden de fuentes tales como las emisiones de los vehículos o polvo procedente de las intrusiones de viento del norte de África (polvo sahariano), frecuente en nuestras latitudes.

El proyecto europeo *Aphesis*, (Alonso, et al 2005), analizaron y consideraron los beneficios que habría si se cumplieran los objetivos fijados por la Directiva 1999/30/CE respecto a las PM. Tras su estudio, obtuvieron que ”Los niveles diarios de PM_{10} por encima de $50\mu g/m^3$ en Bilbao, Madrid y Sevilla son responsables de 1,4 muertes prematuras por 100.000 habitantes y año debido a sus efectos a corto plazo y de 2,8 muertes/100.000 en un periodo de hasta 40 días tras la exposición. A largo plazo, el número de muertes prematuras atribuibles a la contaminación media anual de PM_{10} por encima de $20\mu g/m^3$ es 68/100.000.”. Concluyeron que no se puede obviar el impacto de la contaminación en la salud ya que es perfectamente cuantificable. Controlar los niveles que alcanza la contaminación es un campo de estudio

interesante ya que en él influyen variables tales como la climatología, las medidas realizadas por las estaciones de calidad del aire situadas en la ciudad, tráfico, época del año (aunque la mayor parte de la contaminación sea producida por los transportes, en Madrid aún existen calefacciones muy contaminantes), etc.

Desde hace tiempo la relación entre la contaminación del aire y su influencia como causa de mortalidad es tema de estudio como podemos ver en el proyecto de (Bautista, 2019), donde concluye que "Este estudio describe un aumento de morbimortalidad cardiovascular en España relacionada con las emisiones procedentes de la quema de carbón. Refleja como aquellas comunidades con mayor presencia de centrales térmicas (Asturias y Castilla y León) fueron las más afectadas y extrapola que sus habitantes tienen un mayor riesgo de mortalidad por exposición a partículas $PM_{2,5}$.". O en el artículo de (Alberdi, et al, 1997), donde concluyen que "Las temperaturas frías incrementan la mortalidad para todas las causas y retrasos analizados. El calor produce efectos inmediatos sobre la mortalidad por ACVA en las mujeres.". De estos mismos autores, (Alberdi, et al, 1998) en otro de sus artículos, concluyen que "Los resultados obtenidos sugieren la existencia de una relación causal entre la contaminación atmosférica debida a estos contaminantes y la mortalidad diaria en el Municipio de Madrid".

2.2. Estado del arte

En los últimos años se han llevado a cabo diferentes estudios sobre la contaminación, medidas que se pueden aplicar, control, vigilancia y prevención. En España encontramos diferentes proyectos con múltiples propósitos y campos de estudio relacionados con el tema.

Un ejemplo de ellos es el estudio de (Fernández, 2001) en el que según él mismo "no es más que un esbozo de las posibilidades que este tipo de estructura presenta". En el ártículo sobre las conclusiones del Taller AIRNET de Barcelona, (Baldasano, et al, 2007), analizan exhaustivamente en sus conclusiones la situación actual, y proponen medidas a aplicar para combatir la contaminación.

En la Universidad de Castilla La Mancha (UCLM) se llevó a cabo el proyecto que desembocó en el Sistema integral de calidad del aire que consiste en un sistema de vigilancia, y un sistema de predicción con un día de antelación. Desde la UCLM no le han restado importancia (incluso a los avances conseguidos) y siguen trabajando para mejorar en este campo. Una muestra de ello es la concienciación que quieren implantar en los jóvenes y sembrar en ellos la inquietud por un planeta mejor. Para ello crearon un proyecto sobre la innovación educativa pa-

ra alumnos de secundaria, (Adame, et al, 2005), donde se realizan medidas en una atmósfera próxima a una zona rural y otra industrial. Finalmente se analizan y comparan.

A pesar de los diferentes estudios realizados, los sistemas ya implementados en producción, y algunas medidas implantadas, aún queda mucho camino por andar.

3. Objetivos

3.1. Objetivo general

Con el fin de colaborar a un mejor control y vigilancia de la calidad del aire en Madrid, este trabajo tiene como objetivo construir un modelo de predicción de los niveles de contaminación en Madrid, utilizando técnicas de inteligencia artificial. Además de contar con un sistema de visualización para su mejor gestión.

3.2. Objetivos específicos

Para poder abordar el trabajo se ha dividido el objetivo general en otros más concretos. De esta forma se ha podido realizar una mejor planificación del trabajo. Los objetivos específicos son:

- Capturar datos sobre las medidas de las estaciones de calidad del aire situadas en Madrid. Ha sido necesario obtenerlos de fuentes fiables como el portal de datos abiertos del Ayuntamiento de Madrid (https://datos.madrid.es/portal/site/egob).
- Capturar datos sobre el clima. Esta información la hemos encontrado en la Agencia Estatal de Meteorología, en la sección de datos abiertos (https://www.aemet.es/es/datos_abiertos).
- Analizar, estudiar y tratar los datos. Se han decidido qué datos se van a usar y se han transformado para dejarlos en el formato adecuado para poder aplicar técnicas de inteligencia artificial.
- Aplicar árboles de decisión para poder obtener predicciones mediante la clasificación en diferentes niveles de contaminación preestablecidos.
- Aplicar redes neuronales con el mismo objetivo que los árboles.
- Implementar un entorno de visualización donde el usuario pueda gestionar de forma sencilla los resultados.
- Evaluar los resultados y establecer las conclusiones.
- Proponer posibles mejoras a futuro sobre el trabajo realizado.

4. Metodología del trabajo

El proyecto empieza con la búsqueda de información en la sección *Open Data* del Ayuntamiento de Madrid. Se escogen los *datasets* relacionados con el tema, tanto el histórico de medidas de las estaciones de calidad del aire, como el que corresponde a los valores en tiempo real. Después en la AEMET se obtienen los datos sobre climatología, al igual que en el otro portal se cuenta con un histórico y se prepara la obtención de datos en tiempo real. Todos los conjuntos de datos són públicos.

Una vez adquiridos los datos, se procede a su análisis y estudio. Se procesan para adaptarlos y poder calcular tanto los árboles de decisión como las redes neuronales. Para esta etapa se utiliza como lenguaje de programación Python (Python Software Foundation). Como entorno de trabajo para la programación de código en local se usa Eclipse (Eclipse Foundation).

Para calcular los árboles y las redes neuronales se emplea como conjunto de datos los valores del histórico. Para generarlos y validarlos se usa la librería sklearn (scikit-learn Machine Learning in Python).

Se implementa el *dashboard* donde poder gestionar mediciones y predicciones y se propone un prototipo de visualización mediante Tableau.

Finalmente se evalúan los resultados, se extraen las conclusiones y se proponen las líneas de futuro con las que mejorar el trabajo realizado.

5. Desarrollo específico de la contribución

En este capítulo se explicará en detalle cada una de las fases llevadas a cabo para alcanzar los objetivos específicos comentados previamente. En primer lugar, se explicarán de dónde y cómo se han obtenido los datos. Luego se indicará cómo se han analizado, estudiado y transformado para, finalmente, usarlos para generar árboles y redes neuronales.

5.1. Captura de datos

Los datos que necesitamos obtener para el modelo de predicción, los dividiremos en dos grandes bloques: datos sobre la calidad del aire y datos sobre la climatología. Aunque a continuación explicamos cómo obtenerlos desde las fuentes, es importante resaltar que en el proyecto los descargamos del origen directamente mediante el código de programación de forma automática.

5.1.1. Calidad del aire

Para la obtención de todos los datos sobre la calidad del aire se ha usado la fuente de datos abiertos del Ayuntamiento de Madrid (https://datos.madrid.es/portal/site/egob). Dentro de este portal se han hayado tres *datasets* relevantes para el trabajo:

Calidad del aire. Estaciones de control: este *dataset* se puede descargar en diferentes formatos (XLS, CSV y GEO), nos hemos decantado por el XLS. Nos provee de la estaciones de control de calidad del aire situadas en la ciudad de Madrid. Nos indica por código de estación: la dirección donde se encuentra, altitud, coordenadas geográficas, tipo de estación y magnitudes que mide. A continuación se muestra un extracto (se han reordenado las columnas para que aparezcan las más importantes en la figura 1).

CODIGO	COD_TIPO	NOM_TIPO	NO2	SO2	CO	PM10	PM2_5	O3	BTX	HC	ESTACION	DIRECCION	LONGITUD	LATITUD	ALTITUD	CODIGO_CORTO
28079004	UT	Urbana tráfico	X	X	X						Pza. de España	Plaza de España	-3.7122567	40.4238823	637	4
28079008	UT	Urbana tráfico	X	X	X	X	X	X	X	X	Escuelas Aguirre	Entre C/ Alcalá y C/ O' Donell	-3.6823158	40.4215533	672	8
28079011	UT	Urbana tráfico	X						X		Avda. Ramón y Cajal	Avda. Ramón y Cajal esq. C/ Príncipe de Vergara	-3.6773491	40.4514734	708	11
28079016	UF	Urbana fondo	X		X			X			Arturo Soria	C/ Arturo Soria esq. C/ Vizconde de los Asilos	-3.6392422	40.4400457	695	16
28079017	UF	Urbana fondo	X	X				X			Villaverde	C/. Juan Peñalver	-3.7133167	40.347147	601	17
28079018	UF	Urbana fondo	X	X	X	X		X	X		Farolillo	Calle Farolillo - C/Ervigio	-3.7318356	40.3947825	632	18
28079024	S	Suburbana	X	X	X	X	X	X	X	X	Casa de Campo	Casa de Campo (Terminal del Teleférico)	-3.7473445	40.4193577	646	24
28079027	UF	Urbana fondo	X					X			Barajas Pueblo	C/. Júpiter, 21 (Barajas)	-3.5800258	40.4769179	620	27
28079035	UF	Urbana fondo	X	X	X			X			Pza. del Carmen	Plaza del Carmen esq. Tres Cruces.	-3.7031662	40.4192091	660	35
28079036	UT	Urbana tráfico	X	X	X	X					Moratalaz	Avd. Moratalaz esq. Camino de los Vinateros	-3.6453104	40.4079517	671	36
28079038	UT	Urbana tráfico	X	X		X	X		X		Cuatro Caminos	Avda. Pablo Iglesias esq. C/ Marqués de Lema	-3.7071303	40.4455439	699	38
28079039	UT	Urbana tráfico	X		X			X			Barrio del Pilar	Avd. Betanzos esq. C/ Monforte de Lemos	-3.7115364	40.4782322	676	39
28079040	UF	Urbana fondo	X	X		X					Vallecas	C/ Arroyo del Olivar esq. C/ Río Grande.	-3.6515286	40.3881478	677	40
28079047	UF	Urbana fondo	X			X	X				Mendez Alvaro	C/ Juan de Mariana / Pza. Amanecer Mendez Alvaro	-3.6868138	40.3980991	600	47
28079048	UT	Urbana tráfico	X			X	X				Castellana	C/ Jose Gutierrez Abascal	-3.6903729	40.4398904	680	48
28079049	UF	Urbana fondo	X					X			Parque del Retiro	Paseo Venezuela- Casa de Vacas	-3.6824999999999900	40.41444444444440	662	49
28079050	UT	Urbana tráfico	X			X	X				Plaza Castilla	Plaza Castilla (Canal)	-3.6887449	40.4655841	728	50
28079054	UF	Urbana fondo	X					X			Ensanche de Vallecas	Avda La Gavia / Avda. Las Suertes	-3.6121394	40.3730118	629	54
28079055	UF	Urbana fondo	X			X			X	X	Urb. Embajada	C/ Riaño (Barajas)	-3.5805649	40.4623628	619	55
28079056	UT	Urbana tráfico	X		X	X	X	X			Pza. Elíptica	Pza. Elíptica - Avda. Oporto	-3.7187679	40.3850336	605	56
28079057	UF	Urbana fondo	X	X	X	X					Sanchinarro	C/ Princesa de Eboli esq C/ Maria Tudor	-3.6605173	40.4942012	700	57
28079058	S	Suburbana	X					X			El Pardo	Avda. La Guardia	-3.7746101	40.5180701	612	58
28079059	S	Suburbana	X					X			Juan Carlos I	Parque Juan Carlos I (frente oficinas mantenimiento)	-3.6163407	40.4607255	660	59
28079060	UF	Urbana fondo	X			X		X			Tres Olivos	Plaza Tres Olivos	-3.6897308	40.5005477	715	60

Figura 1: Estaciones de control de calidad del aire en Madrid

Calidad del aire. Datos diarios años 2001 a 2019: este conjunto de datos está formado por múltiples ficheros. Ya que hay un archivo por el histórico de cada año. Y uno en global, en formato DCAT (201410-0-calidad-aire-diario.dcat) en el que están incluidos los enlaces a todos los anteriores. En este caso los *datasets* se pueden obtener en diferentes formatos (TXT, CSV y XML). En nuestro caso hemos optado por trabajar con los CSV. En la figura 2 se muestra un extracto del archivo correspondiente al histórico del año 2019:

```
PROVINCIA;MUNICIPIO;ESTACION;MAGNITUD;PUNTO_MUESTREO;ANO;MES;D01;V01;D02;V02;D03;V03;D04;V04;D05;V05;D06;V06;D07;V07;D08;V08;D09;V09;D10;V10;
28;079;4;1;28079004_1_38;2019;01;00018;V;00020;V;00018;V;00019;V;00018;V;00018;V;00021;V;00020;V;00018;V;00013;V;00016;V;00016;V;00016;V;0002
28;079;4;1;28079004_1_38;2019;02;00013;V;00013;V;00014;V;00018;V;00019;V;00019;V;00019;V;00016;V;00014;V;00015;V;00016;V;00017;V;0001
28;079;4;1;28079004_1_38;2019;03;00018;V;00018;V;00017;V;00017;V;00016;V;00016;V;00014;V;00016;V;00016;V;00017;V;00017;V;00016;V;00017;V;0001
28;079;4;1;28079004_1_38;2019;04;00003;V;00004;V;00004;V;00003;V;00002;V;00002;V;00002;V;00003;V;00003;V;00003;V;00004;V;00003;V;00003;V;0000
28;079;4;6;28079004_6_48;2019;01;000.8;V;001.0;V;000.9;V;001.0;V;000.9;V;001.0;V;001.0;V;001.0;V;000.8;V;000.5;V;000.8;V;000.7;V;000.7;V;001.
28;079;4;6;28079004_6_48;2019;02;000.3;V;000.4;V;000.4;V;000.7;V;000.8;V;000.8;V;000.8;V;000.6;V;000.6;V;000.3;V;000.5;V;000.6;V;000.7;V;000.
28;079;4;6;28079004_6_48;2019;03;000.6;V;000.6;V;000.5;V;000.3;V;000.4;V;000.3;V;000.4;V;000.5;V;000.6;V;000.6;V;000.5;V;000.5;V;000.4;V;000.
28;079;4;6;28079004_6_48;2019;04;000.4;V;000.5;V;000.4;V;000.4;V;000.3;V;000.4;V;000.4;V;000.4;V;000.3;V;000.4;V;000.5;V;000.6;V;000.5;V;000.
28;079;4;7;28079004_7_8;2019;01;00105;V;00163;V;00125;V;00142;V;00108;V;00129;V;00128;V;00157;V;00105;V;00039;V;00104;V;00072;V;00070;V;00210
28;079;4;7;28079004_7_8;2019;02;00005;V;00006;V;00016;V;00079;V;00103;V;00091;V;00103;V;00059;V;00041;V;00005;V;00031;V;00054;V;00079;V;00087
28;079;4;7;28079004_7_8;2019;03;00046;V;00043;V;00023;V;00005;V;00007;V;00006;V;00008;V;00036;V;00043;V;00034;V;00042;V;00031;V;00009;V;00026
28;079;4;7;28079004_7_8;2019;04;00013;V;00025;V;00013;V;00010;V;00007;V;00005;V;00004;V;00006;V;00005;V;00010;V;00026;V;00032;V;00019;V;00006
28;079;4;8;28079004_8_8;2019;01;00071;V;00084;V;00077;V;00086;V;00075;V;00082;V;00087;V;00090;V;00074;V;00051;V;00082;V;00064;V;00062;V;00105
28;079;4;8;28079004_8_8;2019;02;00018;V;00026;V;00041;V;00074;V;00074;V;00066;V;00073;V;00058;V;00054;V;00019;V;00042;V;00058;V;00073;V;00073
28;079;4;8;28079004_8_8;2019;03;00064;V;00058;V;00040;V;00018;V;00033;V;00023;V;00029;V;00052;V;00056;V;00046;V;00050;V;00046;V;00033;V;00052
28;079;4;8;28079004_8_8;2019;04;00045;V;00049;V;00045;V;00033;V;00031;V;00020;V;00019;V;00028;V;00025;V;00038;V;00045;V;00064;V;00054;V;00034
28;079;4;12;28079004_12_8;2019;01;00232;V;00334;V;00269;V;00304;V;00241;V;00279;V;00284;V;00331;V;00236;V;00111;V;00241;V;00174;V;00170;V;004
28;079;4;12;28079004_12_8;2019;02;00026;V;00036;V;00066;V;00195;V;00231;V;00206;V;00230;V;00149;V;00117;V;00027;V;00090;V;00140;V;00195;V;002
28;079;4;12;28079004_12_8;2019;03;00134;V;00125;V;00075;V;00025;V;00045;V;00032;V;00041;V;00108;V;00123;V;00097;V;00114;V;00093;V;00047;V;000
28;079;4;12;28079004_12_8;2019;04;00065;V;00087;V;00064;V;00048;V;00041;V;00027;V;00025;V;00036;V;00033;V;00053;V;00085;V;00113;V;00083;V;000
28;079;8;1;28079008_1_38;2019;01;00012;V;00014;V;00012;V;00013;V;00013;V;00018;V;00016;V;00014;V;00013;V;00011;V;00012;V;00011;V;00011;V;0001
28;079;8;1;28079008_1_38;2019;02;00007;V;00009;V;00010;V;00010;V;00012;V;00010;V;00011;V;00009;V;00009;V;00007;V;00010;V;00010;V;00011;V;0001
28;079;8;1;28079008_1_38;2019;03;00009;V;00009;V;00009;V;00008;V;00008;V;00008;V;00008;V;00008;V;00009;V;00009;V;00010;V;00009;V;00010;V;0000
28;079;8;1;28079008_1_38;2019;04;00008;V;00008;V;00009;V;00009;V;00008;V;00008;V;00008;V;00008;V;00009;V;00010;V;00009;V;00009;V;0000
28;079;8;6;28079008_6_48;2019;01;000.6;V;000.7;V;000.3;V;000.5;V;000.6;V;000.8;V;000.5;V;000.4;V;000.2;V;000.3;V;000.3;V;000.4;V;001.
```

Figura 2: Datos diarios de la calidad del aire en 2019. Histórico

El archivo cuenta con cabecera por lo que se pueden intuir los valores, vendrían en orden de izquierda a derecha (ejemplo con la primera línea): provincia [28], municipio [079], estación [4], magnitud [1], punto de muestreo [28079004_1_38], año [2019], mes [01], D01 (valor obtenido el día 1) [00018], V01 (en caso de ser V el registro previo es válido, en caso de ser N sería inválido) [V], D02 (valor obtenido el día 2) [00020], V02 (validez del registro del día 2) [V], y sucesivamente para cada día del mes. Siempre llega hasta 31, aunque obviamente en los

meses que no tienen tantos días lo registros de esos días serán invalidados mediante N en el campo de validación VXX. Para mayor detalle sobre la comprensión del *dataset* se puede consultar la documentación asociada (enlace a calidad_del_aire_global.pdf) que proporciona la propia web.

Para este punto existe otra opción que es descargar los datos históricos en vez de por día por horas. Pero se ha escogido la opción diaria por simplicidad a la hora de tratar los datos. Este *dataset* forma parte del conjunto de datos que se utiliza para entrenar los modelos.

Calidad del aire. Datos en tiempo real: contiene los datos de medición de las estaciones en tiempo real. Se actualiza cada hora entre los primeros 20 y 30 minutos. Es el *dataset* que usaremos reiteradamente para obtener los datos en cada momento y los que introduciremos en los modelos generados para obtener predicciones. Al igual que el caso anterior, se puede descargar en XML, CSV y TXT. Los datos en tiempo real se van a guardar para su posterior procesado en TXT. En la figura 3 podemos observar un extracto del conjunto de datos de calidad del aire en tiempo real:

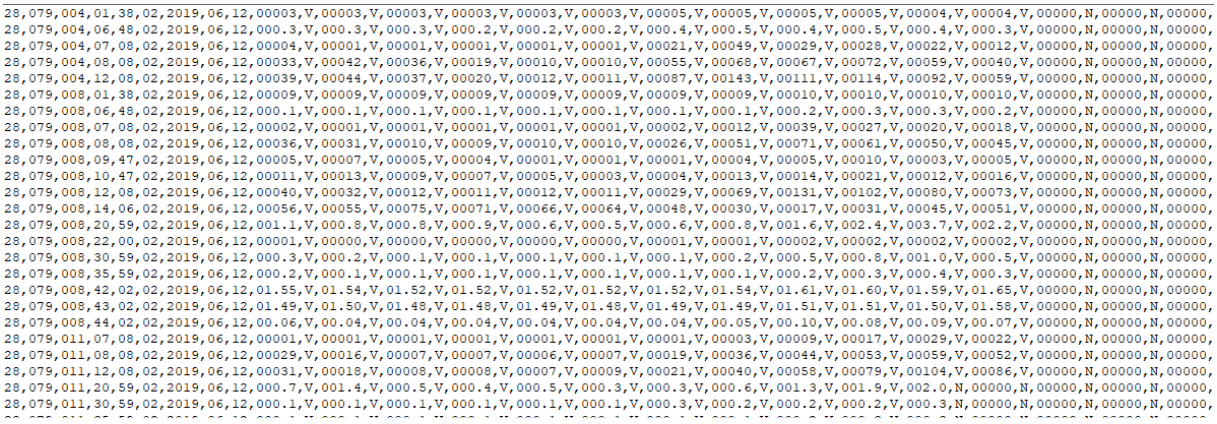


Figura 3: Datos por hora de la calidad del aire en tiempo real

En este caso el archivo no cuenta con cabecera. Pero la estructura es similar al conjunto de datos histórico. Los campos serían (ejemplo con la primera fila): provincia [28], municipio [079], estación [004], magnitud [01], técnica [38], periodo de análisis [02], año [2019], mes [06], día [12], dato H01 (valor obtenido la primera hora del día) [00003], código de validación V01 (en caso de ser V el registro previo es válido, en caso de ser N sería inválido) [V], dato H02 (valor obtenido en la segunda hora del día) [00003], código de validación V02 (validez del registro de la hora 2) [V], y así sucesivamente para cada hora del día. Por defecto el *dataset* del día tiene todas las columnas referentes a las 24 horas. Los valores de las horas que aún no han llegado, se marcan con el código de validación N, indicando así la invalidez del registro previo. Al igual

que en el caso anterior, para más información sobres los campos se puede consultar la documentación asociada que contiene todos los detalles (enlace a calidad_del_aire_global.pdf).

5.1.2. Climatología

De igual manera que con los datos de medición de calidad del aire, para la climatología necesitamos datos históricos para poder llevar a cabo la generación y entrenamiento de los modelos de predicción, como datos en tiempo real. La idea inicial es extraer todos los datos de la sección de datos abiertos de la AEMET (https://opendata.aemet.es/dist/index.html).

Datos climatológicos históricos: al intentar localizar y descargar el conjunto de datos con el histórico nos encontramos con un problema. La AEMET *OpenData*, el API REST desde el cual se pueden descargar los datos gratuitamente. A priori, en la documentación (https://opendata.aemet.es/dist/index.html), tiene dos consultas que podrían ser válidas ya que parece que en ellas te devuelve la climatología diaria por estación, o la de todas las estaciones. Las podemos ver en la figura 4:

Figura 4: Consultas climatología diaria en la AEMET *Open Data*

Cuando intentamos hacer una consulta, con un rango temporal al que cubren los datos históricos de calidad del aire (años), nos devuelve el mensaje de que el intervalo de tiempo como máximo tiene que ser de 31 días. Aunque da la consulta por correcta, al devolvernos el código 200, a su vez en el cuerpo nos indica el error 404 de que la petición ha sido devuelta sin datos. En la figura 5 se observan los detalles.

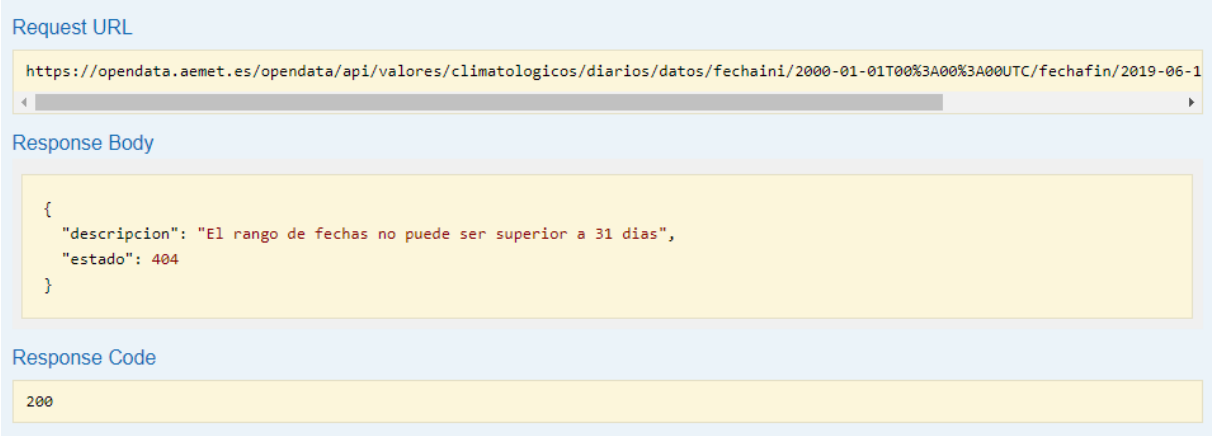


Figura 5: Consultas climatología diaria de todas las estaciones en la AEMET *Open Data*

Por lo que lo intentamos con la otra consulta, especificando por estación concreta. En Madrid, contamos con dos, una en Ciudad Universitaria (código 3194U) y otra en el Retiro (código 3195). Para el trabajo hemos utilizado únicamente los datos de la estación de Retiro, unificando y simplificando la primera versión del proyecto. De la misma manera que en la anterior consulta intentamos cubrir todo el rango temporal, obteniendo como resultado un mensaje de error similar al anterior en el que el rango máximo es de 5 años (figura 6)

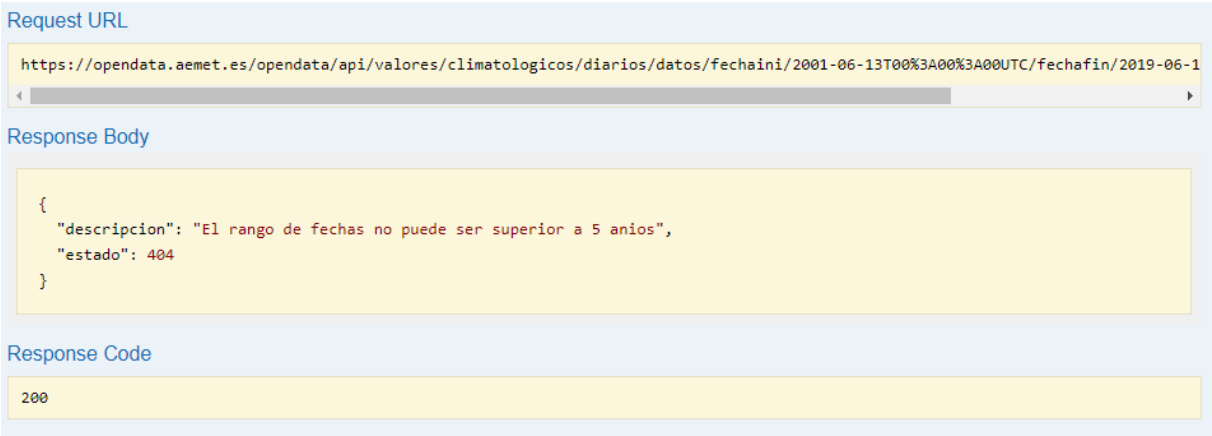


Figura 6: Consultas climatología diaria de la estación de Retiro en la AEMET *Open Data*

Finalmente descargamos 4 ficheros para cubrir el rango temporal con el que contamos en datos de calidad del aire. Desde enero de 2001 hasta mayo de 2019.

Las respuestas de la consultas nos proporcionan dos URL cada uno para la descarga de dos ficheros JSON, el primero contiene los metadatos y el segundo los datos. En la figura 7 se muestra el primer objeto de uno de los archivos descargados. La interpretación es bastante intuitiva ya que los nombres de los atributos son similares en la vida real.

Figura 7: Objeto JSON devuelto en los datos de la consulta climatología diaria de la estación de Retiro en la AEMET *Open Data*

Datos climatológicos en tiempo real: los datos climatológicos referentes al día, usados como *input* en los modelos entrenados para poder obtener una predicción del nivel de contaminación los conseguiremos a través de otra consulta (Figura 8) de la AEMET.

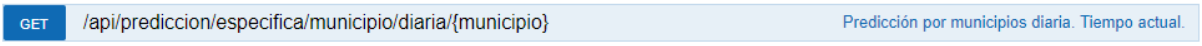
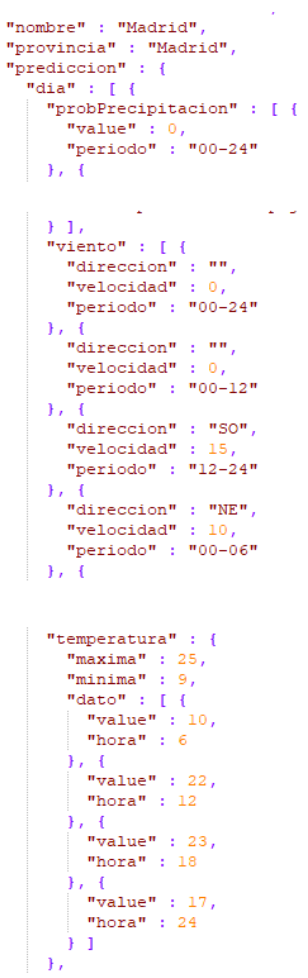


Figura 8: Consulta específica de la climatología diaria por municipio en la AEMET *Open Data*

Ídem a la respuesta de la consulta del histórico de la climatología, el resultado son dos ficheros JSON, uno con los metadatos y otro con los datos. En la figura 9 podemos ver en un collage del archivo, la estructura de sus objetos, para hacernos una idea en global.



```
"nombre" : "Madrid",
"provincia" : "Madrid",
"prediccion" : {
  "dia" : [ {
    "probPrecipitacion" : [ {
      "value" : 0,
      "periodo" : "00-24"
    }, {
    } ],
    "viento" : [ {
      "direccion" : "",
      "velocidad" : 0,
      "periodo" : "00-24"
    }, {
      "direccion" : "",
      "velocidad" : 0,
      "periodo" : "00-12"
    }, {
      "direccion" : "SO",
      "velocidad" : 15,
      "periodo" : "12-24"
    }, {
      "direccion" : "NE",
      "velocidad" : 10,
      "periodo" : "00-06"
    }, {

    "temperatura" : {
      "maxima" : 25,
      "minima" : 9,
      "dato" : [ {
        "value" : 10,
        "hora" : 6
      }, {
        "value" : 22,
        "hora" : 12
      }, {
        "value" : 23,
        "hora" : 18
      }, {
        "value" : 17,
        "hora" : 24
      } ]
    },
```


Figura 9: Extracto del objeto JSON devuelto en los datos de la consulta específica diaria por municipio en la AEMET *Open Data*

En este momento nos encontramos con una discrepancia. El objeto JSON de la consulta sobre el histórico de la climatología, las precipitaciones toman el valor de los mm. Por lo que el modelo se va a entrenar con la precipitación en unidades de mm. Mientras que en los datos obtenidos diariamente con esta última consulta que comentábamos, las precipitaciones no se miden igual, sino el valor que obtenemos de la consulta es el porcentaje de probabilidad de que llueva. Por lo que no se puede usar este dato como *input* a un modelo entrenado con la otra unidad. No se ha hallado consulta en la AEMET *Open Data* que nos proporcione el valor de la precipitación para el día actual en mm (usar la consulta del histórico con fecha de ”hoy” no es una opción porque la periodicidad de actualización es de 1 vez al día, con un retardo de 4 días). Como alternativa, mediante la técnica de *web scrapping* con Python, obtenemos ese valor desde la web https://www.eltiempo.es/.

Añadir como información adicional que para la descarga de datos de la sección *open*

data de la AEMET es necesario contar con un API KEY, que se puede solicitar gratuitamente en este enlace https://opendata.aemet.es/centrodedescargas/altaUsuario?.

5.2. Tecnología

En este trabajo se ha usado Python como lenguaje de programación. Como entorno de trabajo se ha empleado Eclipse. El *dashboard* se ha creado con Tableau. A continuación, se profundiza en estas tecnologías.

5.2.1. Lenguaje de programación Python

Figura 10: Logo de Python

Python es un lenguaje de programación creado en 1991 por Guido Van Rossum, en Holanda. Posee una licencia de código abierto, denominada Python Software Foundation License. El propósito de Python es ser un lenguaje de programación orientado a objetos y preparado para realizar cualquier tipo de programa, desde aplicaciones a servidores de red o páginas web.

Dentro de sus características más importantes resaltaríamos que es un lenguaje multiplataforma, es decir, hay versiones disponibles de Python en muchos sistemas operativos diferentes. Incluso cualquier sistema es compatible con el lenguaje siempre que exista un intérprete programado para él. Es interactivo ya que dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede sernos útil para entender mejor y más rápido el lenguaje. Es orientado a objetos, por lo que ofrece una manera sencilla de crear programas con componentes reutiliza-

bles. Otra ventaja de Python es su sintaxis, porque es muy visual gracias a una notación con márgenes (identado) frente a otros lenguajes en el que para separar porciones de código se utilizan elementos como las llaves o palabras clave tales como *begin* y *end*.

Respecto a la dificultad de aprendizaje, si el programador cuenta con algo de experiencia en otros lenguajes probablemente sea muy sencillo de aprender. En cualquier caso no tener experiencia no es un problema, ya que en la propia web de Python (https://www.python.org/) se pueden encontrar información y tutoriales detallados para aprender Python desde cero. Además de una guía completa para principiantes.

Una de las ventajas indiscutibles de Python frente a otros lenguajes de programación es que es simplificado y rápido por lo que hace la programación más sencilla. Es portable al ser multiplataforma. Cuenta con una comunidad que cuida el lenguaje y lo van actualizando. Aunque como nada es perfecto, tiene alguna desventaja y es que la curva de aprendizaje cuando ya estás en la parte web no es tan sencilla, ya que muchas de las utilidades están hechas por terceros y no siempre funcionan como uno desearía.

Para nuestro proyecto hemos escogido Python porque es una opción interesante para realizar el programa y que éste se pueda ejecutar en cualquier máquina. Es gratuito y de alto nivel. Cuenta con múltiples librerías por lo que la implementación a priori no es muy complicada, permitiendo desarrollar en poco tiempo prototipos *software*. En el trabajo se han usado diferentes librerías preinstaladas con Python, se han tenido que añadir algunas extra, pero se querría hacer una mención especial a la librería utilizada para la parte de inteligencia artificial (árboles y redes neuronales en nuestro caso), scikit-learn (https://scikit-learn.org/)

Para poder usar Python, hay que descargarlo desde su web (https://www.python.org). El *software* de este trabajo se ha desarrollado con la última versión de Python disponible en este momento, Python 3.7.3. El instalador que descargamos tiene un propio asistente para la instalación, por lo que es sencillo.

5.2.2. Entorno de desarrollo Eclipse

Figura 11: Logo de Eclipse

Eclipse es un entorno de desarrollo que nos proporciona las herramientas para poder programar el código fuente. Eclipse va un paso más allá siento un IDE, es decir, un entorno de desarrollo integrado que se puede usar tanto para escribir, generar, probar y depurar un programa. Cuenta con una interfaz de usuario para programar y depurar en diferentes modos. Eclipse está desarrollado en Java, por lo que es multiplataforma además de contar con una interfaz amigable y práctica a la hora del trabajo.

Inicialmente Eclipse fue desarrollado por IBM, aunque ahora es gestionado por una fundación independiente sin ánimo de lucro, Eclipse Foundation. Es gratuito y cuenta con una gran comunidad que lo usa, por lo que existe mucha información al respecto y foros donde poder acudir en caso de duda. La fundación Eclipse fomenta el código abierto por lo que permite la instalación de extensiones y plugins para poder ir aumentando la funcionalidad del entorno.

Aunque Eclipse quizás inicialmente estuviera más enfocado para desarrollar Java, gracias a la comunidad que lo rodea y las extensiones, soporta diferentes lenguajes de programación, como PHP, C, Python, etc.

La última versión Eclipse, a fecha de hoy, IDE 2019-06 se puede descargar desde la página web de eclipse.org. Esta versión cuenta con el soporte de idiomas políglota para Java, JavaScript, C, C++, PHP, Python, etc. Ha mejorado el tiempo de arranque frente a otras versiones. Tiene una sólida infraestructura para soportar la integración de servidores de idiomas. Cuenta con una gran variedad de extensiones y complementos para la plataforma ya probados. Es necesario tener instalado Java en el ordenador para poder ejecutar el entorno.

Para el trabajo nos hemos decantado por este entorno porque estamos previamente familiarizados con él. Es gratuito y fiable.

5.2.3. Tableau Software

Figura 12: Logo de Tableau Software

Tableau es una herramienta de visualización de datos utilizada normalmente en el área de *Business Intelligence*. Ayuda al usuario a ver y comprender los datos. Las funcionalidades más interesantes de Tableau son: permite diferentes conexiones de datos (sql server, cloudera, MongoDB, Dropbox, etc), se pueden crear cuadros de mandos informativos e interactivos, detección de datos rápida y fácil, etc.

Para el proyecto vamos a utilizar Tableau Desktop, en su versión completa, no es gratuito. Pero permite una prueba de 14 días con el *software* completo, o se puede conseguir una licencia de estudiante durante un año (es la licencia con la que se cuenta en este momento). En el caso de tener la versión gratuita, faltarían algunas funcionalidades como las de guardar en local las visualizaciones, pero esto se puede suplir mediante Tableu Public. Es decir, creándonos una cuenta en la nube de Tableau, las visualizaciones realizadas las podremos publicar en internet y de esta forma no las perderemos.

Para poder instalar el *software* hay que descargarlo directamente desde la página de Tableau (https://www.tableau.com/).

Para crear el dasboard hemos escogido Tableau, porque a pesar de que no es completamente gratuito, tiene mucho potencial. Es sencillo de utilizar y se consiguen buenos resultados. La aplicación desarrollada exporta los datos resultado en archivos CSV y el *dashboard* creado en Tableau actualiza los datos automáticamente según se actualizan en las tablas CSV ya que está conectado.

5.3. Estudio y transformación de los datos

Una vez que ya disponemos de las fuentes de datos y tenemos claro los *datasets* a usar así como la tecnología, el siguiente paso es el estudio de la información con la que contamos, para procesarla y transformarla en datos útiles.

***Dataset* Calidad del aire. Estaciones de control**

El primer conjunto de datos que se va a estudiar es el que contempla las estaciones de calidad del aire, su ubicación y magnitudes que mide cada una de ellas.

De los campos con los que cuenta el conjunto de datos, no nos van a ser útiles todos, a continuación en la figura 13, se muestra el *dataset* ocultando los datos que no son de nuestro interés para una mejor comprensión:

- CODIGO: código identificador único de cada estación.
- NO2, SO2, CO, PM10, PM2_5 y O3: dióxido de carbono, dióxido de sodio, monóxido de carbono, partículas por millón en suspensión menor de 10 microgramos, partículas por millón en suspensión menor de 2.5 microgramos y ozono, respectivamente. La estación en la que tenga una X en cualquiera de estas magnitudes indica que la mide.
- LONGITUD y LATITUD: ubicación de cada estación referenciada por las coordenadas de longitud y latitud.

	A	J	K	L	M	N	O	Y	Z
1	CODIGO	NO2	SO2	CO	PM10	PM2_5	O3	LONGITUD	LATITUD
2	28079004	X	X	X				-3.7122567	40.4238823
3	28079008	X	X	X	X	X	X	-3.6823158	40.4215533
4	28079011	X						-3.6773491	40.4514734
5	28079016	X		X			X	-3.6392422	40.4400457
6	28079017	X	X				X	-3.7133167	40.347147
7	28079018	X	X	X	X		X	-3.7318356	40.3947825
8	28079024	X	X	X	X	X	X	-3.7473445	40.4193577
9	28079027	X					X	-3.5800258	40.4769179
10	28079035	X	X	X			X	-3.7031662	40.4192091
11	28079036	X	X	X	X			-3.6453104	40.4079517
12	28079038	X	X		X	X		-3.7071303	40.4455439
13	28079039	X		X			X	-3.7115364	40.4782322
14	28079040	X	X		X			-3.6515286	40.3881478
15	28079047	X			X	X		-3.6868138	40.3980991
16	28079048	X			X	X		-3.6903729	40.4398904
17	28079049	X					X	-3.6824999999999900	40.41444444444440
18	28079050	X			X	X		-3.6887449	40.4655841
19	28079054	X					X	-3.6121394	40.3730118
20	28079055	X			X			-3.5805649	40.4623628
21	28079056	X		X	X	X	X	-3.7187679	40.3850336
22	28079057	X	X	X	X			-3.6605173	40.4942012
23	28079058	X					X	-3.7746101	40.5180701
24	28079059	X					X	-3.6163407	40.4607255
25	28079060	X			X		X	-3.6897308	40.5005477

Figura 13: *Dataset* Calidad del aire. Estaciones de control. Datos últiles

Para las descargas de ficheros en general se ha programado una función a la que únicamente hay que indicarle como parámetro el URL de descarga, la ubicación donde se desea guardar y la acción, es decir, en caso de que existiera el archivo si se quiere guardar con otro nombre, si se prefiere reemplazar o si no se quiere hacer nada. En el fragmento de código 1 se muestra la cabecera de la función.

```
1 def descargarArchivosUrl (urlDescargar=None, urlGuardar=None, accion 0):
2     """Descarga un archivo de la web mediante la url y lo guarda en disco.
3
4     Devuelve True si se ha descargado y guardado el archivo correctamente, sino
          devolvera False.
5
6     Parametros:
7     urlDescargar -- url del archivo a descargar
8     urlGuardar -- ruta donde se desea guardar el dataset (incluido el nombre
```

```
        del fichero)
9     accion -- 0 = reemplazar el archivo si existe, 1 = guardar con otro nombre
        sin reemplazar (nombre (n), n=1,2,3...), 2 = no sustituir ni guardar el
        archivo si existe.
10
11    Excepciones:
12    ValueError -- Si (urlDescargar = None) o (urlGuardar = None)
13    """
14
15    ...
```

Fragmento de código 1: Función descargarArchivosUrl de datos_web.py

El fichero se obtiene mediante la función. Se llama a la función anterior indicándole los URL de descarga y guardado. Siempre se va a sobre escribir el archivo por si hubiera actualizaciones. En el fragmento de código 2 podemos ver la función a la que nos referimos.

```
1  def get_estaciones_control ():
2      """Descarga el dataset con la estaciones de control de aire, su ubicacion y
         las magnitudes que mide cada una. Lo guarda en disco.
3
4      Excepciones:
5      ValueError -- Si la funcion descargarArchivosUrl propaga algun error.
6
7      """
8
9      urlDescargar = "https://datos.madrid.es/egob/catalogo/212629-0-estaciones-
10     control-aire.xls"
11     urlGuardar = "../data/estaciones_control/212629-0-estaciones-control-
12     aire.xls"
13
14     ...
```

Fragmento de código 2: Función get_estaciones_control de datasets.py

Después de la descarga los datos son procesados para su posterior uso. Se lleva a cabo mediante la función del fragmento de código 3:

```
def tratar_dataset_estaciones_control():
    """Procesa el dataset de la estaciones de control.
    Crea una lista con diccionarios donde cada diccionario son los datos de una
        estacion.

    Devuelve un objeto lista con los diccionarios con la informacion sobre las
        estaciones

    """

    ...
```

Fragmento de código 3: Función tratar_dataset_estaciones_control de datasets.py

***Dataset* Calidad del aire. Datos diarios años 2001 a 2019**

De los conjuntos de datos con los valores históricos de la calidad del aire vamos a utilizar todos los campos a priori, salvo los cuatro primeros de PROVINCIA, MUNICIPIO, ESTACIÓN y MAGNITUD. Ya que procesando el PUNTO DE MUESTREO podemos extraer todos los datos anteriores. La vista del *dataset* en CSV abierto con Excel y ocultando las columnas que no nos aplican, tendría la forma de la figura 14.

	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
1	PUNTO_MUESTREO	ANO	MES	D01	V01	D02	V02	D03	V03	D04	V04	D05	V05	D06	V06	D07	V07	D08	V08	D09	V09	D10	V10
2	28079004_1_38	2019	1	18	V	20	V	18	V	19	V	18	V	18	V	21	V	20	V	18	V	13	V
3	28079004_1_38	2019	2	13	V	13	V	14	V	18	V	19	V	19	V	19	V	16	V	16	V	14	V
4	28079004_1_38	2019	3	18	V	18	V	17	V	17	V	16	V	16	V	14	V	16	V	16	V	17	V
5	28079004_1_38	2019	4	3	V	4	V	4	V	3	V	2	V	2	V	2	V	3	V	3	V	3	V
6	28079004_6_48	2019	1	000.8	V	001.0	V	000.9	V	001.0	V	000.9	V	001.0	V	001.0	V	001.0	V	000.8	V	000.5	V
7	28079004_6_48	2019	2	000.3	V	000.4	V	000.4	V	000.7	V	000.8	V	000.8	V	000.8	V	000.6	V	000.6	V	000.3	V
8	28079004_6_48	2019	3	000.6	V	000.6	V	000.5	V	000.3	V	000.4	V	000.3	V	000.4	V	000.5	V	000.6	V	000.6	V
9	28079004_6_48	2019	4	000.4	V	000.5	V	000.4	V	000.4	V	000.3	V	000.4	V	000.4	V	000.4	V	000.3	V	000.4	V
10	28079004_7_8	2019	1	105	V	163	V	125	V	142	V	108	V	129	V	128	V	157	V	105	V	39	V
11	28079004_7_8	2019	2	5	V	6	V	16	V	79	V	103	V	91	V	103	V	59	V	41	V	5	V
12	28079004_7_8	2019	3	46	V	43	V	23	V	5	V	7	V	6	V	8	V	36	V	43	V	34	V
13	28079004_7_8	2019	4	13	V	25	V	13	V	10	V	7	V	5	V	4	V	6	V	5	V	10	V
14	28079004_8_8	2019	1	71	V	84	V	77	V	86	V	75	V	82	V	87	V	90	V	74	V	51	V
15	28079004_8_8	2019	2	18	V	26	V	41	V	74	V	74	V	66	V	73	V	58	V	54	V	19	V

Figura 14: *Dataset* Calidad del aire. Datos diarios años 2001 a 2019. Datos últiles

El significado de cada campo es:

- PUNTO DE MUESTREO: formado por <CODIGO DE LA ESTACIÓN>, <MAGNITUD>y <TÉCNICA>. De este campo extraeremos tanto el código de la estación como la magnitud.
- AÑO: año de la medida
- MES: mes de la medida
- DXX: valor que tomó dicha magnitud el día XX del mes.
- VXX: si es V, el valor del día XX sería correcto. Si es N, el valor de dicho día no sería válido.

En este caso mediante la función del fragmento de código 4, se decargará el DCAT del que se crea una lista con todos los URL de los archivos históricos. Finalmente, para cada uno de ellos se llama a la función *descargarArchivosUrl* y se guardan para su posterior tratamiento. Los datos cubren desde el año 2001 hasta la actualidad. Se descargan todos, aunque en el futuro pueda no usarse todos los datos por la limitación en los datos de climatología.

```
1  def get_historicos ():
2      """Descarga los ficheros con los datos historicos de calidad del aire desde
           el anio 2001 hasta la actualidad
3      Los guarda en disco.
4
5      Devuelve una lista con las ubicaciones de todos los datasets guardados en
           local
6
7      Excepciones:
8      ValueError -- Si la funcion descargarArchivosUrl propaga algun error.
9
10     """
11
12     ...
```

Fragmento de código 4: Función get_historicos de datasets.py

Dataset Datos climatológicos históricos

De los objetos del fichero JSON descargado utilizaremos los atributos:

- "fecha": fecha de los valores que vienen a continuación
- "tmed": temperatura media del día
- "prec": precipitación diaria de 07 a 07 medidas en mm
- "velmedia": velocidad media del viento

Para la descarga se utilizará la función del fragmento de código 5.

```
def get_climatologia_historico():
    """Descarga el fichero JSON con los datos historicos de la climatologia de
       Madrid desde el anio 2001
    hasta la actualidad. Lo guarda en disco.

    Excepciones:
    ValueError -- Si la funcion descargarArchivosUrl propaga algun error.

    """

    api_key = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJsdXptYXJpYS5tdHRAZ21haWwuY
    29tIiwianRpIjoiM2VhNDVkZTgtYjFhZS00Y2M1LTllY2UtYjZmZDgwZGI4OGEyIiwiaX
    NzIjoiQUVNRVQiLCJpYXQiOjE1NjAzMzg5NjYsInVzZXJJZCI6IjNlYTQ1ZGU4LWIxYWU
    tNGNjNS05ZWNlLWI2ZmQ4MGRiODhhMiIsInJvbGUiOiIifQ.TxnmkwDmYfXhQFPjVE_nE
    uEd5fcWugbenooNuypGX3w"
    url = "https://opendata.aemet.es/opendata/api/valores/climatologicos/
    diarios/datos/fechaini/2014-06-01T00%3A00%3A00UTC/fechafin/2019-05-31
    T23%3A59%3A59UTC/estacion/3195"

    ...
```

Fragmento de código 5: Función get_climatologia_historico de datasets.py

Una vez tenemos los datos guardados. Se procesan mediante la función del fragmento de código 6. Se guardan en nuevo fichero denominado clima_hist.csv con los datos procesados

para su posterior uso. Podemos ver el aspecto del archivo resultante visto desde Excel en la figura 15.

```
1 def tratar_dataset_climatologia_historico ():
2     """Procesa el dataset historico de la climatologia.
3     Crea una nuevo fichero llamado clima_hist.csv con los datos relevantes.
4     Deja los registros ordenados por fecha, desde el mas actual al mas antiguo.
5
6     """
7
8     ...
```

Fragmento de código 6: Función tratar_dataset_climatologia_historico de datasets.py

	A	B	C	D	E	F
1	anio	mes	dia	temperatura	viento	precipitacion
2	2019	5	31	24.2	14.0	0.0
3	2019	5	30	21.7	17.0	0.0
4	2019	5	29	18.4	19.0	0.0
5	2019	5	28	19.8	25.0	0.0
6	2019	5	27	21.7	18.0	0.0
7	2019	5	26	19.0	21.0	0.0
8	2019	5	25	18.5	18.0	0.0
9	2019	5	24	20.3	22.0	0.0
10	2019	5	23	21.6	17.0	0.0
11	2019	5	22	20.7	15.0	0.0
12	2019	5	21	19.4	14.0	0.0
13	2019	5	20	15.3	14.0	0.0

Figura 15: *Dataset* procesado Datos climatológicos históricos. Datos últiles

Procesado de los datos históricos

Ya hemos obtenido tanto los datos históricos de las estaciones de calidad del aire como los datos históricos del clima. Por lo que ahora se van a procesar y unificar. La función del fragmento de código 7 nos permitirá unificar los datos y a su vez separarlos por estaciones. Es decir, como resultado generará una carpeta llamada entrenamiento en la que habrá por cada

código de estación una carpeta, en la que se incluirá un archivo llamado data_totest_<código estación>.csv. En la figura 16 podemos observar la estructura del archivo para la estación 28079024.

```
1 def tratar_dataset_historicos_training():
2     """Procesa los datasets resultados de tratar los datos historicos. Unifica
          tanto las medidas de las magnitudes como los valores del clima.
3     Genera una carpeta llamada entrenamiento en la que dentro se hayara una
          carpeta con el codigo de cada estacion que a su vez contendra un
4     fichero con los datos historicos completos de esa estacion.
5     Preparados para su posterior procesado como datos de entrenamiento.
6
7     """
8
9     ...
```

Fragmento de código 7: Función tratar_dataset_historicos_training de datasets.py

	A	B	C	D	E	F	G	H	I	J
1	so2	co	no2	pm10	pm2_5	o3	temp	viento	precipitacion	n_predicion
2	4.0	0.2	20.0	7.0	11.0	65.0	19.7	15.0	0.0	NaN
3	4.0	0.2	19.0	6.0	9.0	72.0	19.2	15.0	0.0	NaN
4	4.0	0.2	16.0	7.0	9.0	72.0	17.6	15.0	0.0	NaN
5	4.0	0.2	5.0	3.0	4.0	80.0	12.1	16.0	0.0	0
6	4.0	0.2	4.0	3.0	5.0	81.0	10.0	26.0	5.9	0
7	4.0	0.2	14.0	2.0	3.0	67.0	8.0	23.0	29.4	0
8	4.0	0.2	6.0	6.0	8.0	95.0	10.0	27.0	2.9	0
9	4.0	0.2	12.0	9.0	15.0	88.0	13.5	12.0	0.0	0
10	4.0	0.2	6.0	5.0	7.0	102.0	10.6	23.0	0.1	0
11	4.0	0.2	7.0	8.0	13.0	99.0	14.2	24.0	0.3	0
12	4.0	0.2	13.0	3.0	3.0	76.0	9.9	12.0	7.6	0
13	4.0	0.2	11.0	4.0	4.0	91.0	11.6	14.0	27.6	0
14	4.0	0.2	13.0	11.0	24.0	85.0	16.9	19.0	1.7	0
15	4.0	0.2	19.0	4.0	8.0	83.0	14.8	14.0	0.0	0
16	5.0	0.2	17.0	9.0	15.0	52.0	18.5	23.0	0.0	0

Figura 16: Datos históricos procesados completos para la estación 28079024

Si observamos los datos de la figura 16, podemos ver aparte de los valores de las diferentes magnitudes que mide esta estación y los valores del clima, un nuevo campo llamado n_prediccion. Será el valor que ha tomado ese registro respecto a los niveles de predicción que se han marcado para este trabajo (Cuadro 2).

NIVEL	RANGO DEL VALOR DE NO_2
0	De 0 a 49 $\mu g/m^3$
1	De 50 a 99 $\mu g/m^3$
2	De 100 a 149 $\mu g/m^3$
3	Mayor de 150 $\mu g/m^3$

Cuadro 2: Rango de valores de NO_2 al que pertenece cada nivel

El cálculo se ha realizado de la siguiente forma: por ejemplo, para un día x se ha observado el valor de NO_2 que había después de 3 días. Dependiendo del rango en el que se encontrara el campo n_prediccion de su fila de registro toma el valor del nivel perteneciente a ese rango. Es la razón por la que los tres últimos días de la tabla (recordemos que se ordenaba desde el día más actual al más antiguo), es decir, los tres días más recientes están como NaN, ya que no hay datos posteriores para poder calcular el n_prediccion.

Como último paso antes de pasar a la inteligencia artificial con lo que a los datos históricos se refiere. Se hará un procesado final en el que se eliminarán los registros con datos nulos (NaN). Y se les darán formato por estación. Se generarán dos archivos (no es estrictamente necesario guardarlos en ficheros, ya que los datos se podrían pasar directamente a los modelos de predicción, pero para poder hacer un seguimiento más detallado de las transformaciones llevadas a cabo, se hace de esta manera), uno como *input* del árbol (data_tree_<código estación>.txt) y otro de la red neuronal (data_neuronal_network_<código estación>.txt). En nuestro caso, ambos archivos son iguales. Pero se ha decidido implementarlo como dos diferentes por si el día de mañana los atributos no fueran los mismos para cada modelo. La función empleada es la del fragmento de código 8.

```
1 def crear_input_clasificadores_dataset ():
2     """Procesado final en el que se eliminan los registros nulos y se da
          formato a los datos por cada estacion.
3     Partiendo del archivo de cada estacion data_totest_<estacion>.csv se
          formatearan los datos y se guardaran en dos ficheros, uno para el arbol
          y el otro para la red neuronal (data_tree_<estacion>.txt y
          data_neuronal_network_<estaccion>.txt)
4
5     """
6
```

```
    ...
```

Fragmento de código 8: Función crear_input_clasificadores_dataset de datasets.py

A continuación en la figura 17, podemos ver el resultado del fichero data_tree_28079024.txt de la estación 28079024. En la primera línea contamos con una lista con los atributos (clase - n_prediccion incluida). En la segunda línea estarían los valores correspondientes al atributo n_prediccion separado por ",". Mientras que en la tercera línea se encuentran los conjuntos de valores del resto de atributos para cada registro. Los valores dentro de un conjunto están separados por ",", mientras que los conjuntos en si están separados por ",,".

```
1  ['so2', 'co', 'no2', 'pm10', 'pm2_5', 'o3', 'temp', 'viento', 'precipitacion', 'n_predicion']
2  0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
3  4.0,0.2,5.0,3.0,4.0,80.0,12.1,16.0,0.0,,4.0,0.2,4.0,3.0,5.0,81.0,10.0,26.0,5.9,,4.0,0.2,14.0,2.0,3.0,67.0,8.0,23.0,29.4,,4
```

Figura 17: Datos tras el procesado final para la estación 28079024

***Dataset* Calidad del aire. Datos en tiempo real**

Nos descargamos el conjunto de datos con los valores de las estaciones que han medido en tiempo real. Lo conseguimos mediante la función del fragmento de código 9.

```
def get_tiempo_real ():
    """Descarga el dataset con la informacion de las medidas realizadas por la
       estaciones (actualizaciones cda hora). Lo guarda en disco.

    Excepciones:
    ValueError -- Si la funcion descargarArchivosUrl propaga algun error.

    """

    urlDescargar = 'https://datos.madrid.es/egob/catalogo/212531-0-calidad-aire-
    tiempo-real.dcat'
    urlGuardar = "../data/tiempo_real/212531-0-calidad-aire-tiempo-
    real.dcat"

    ...
```

Fragmento de código 9: Función get_tiempo_real de datasets.py

En la figura 18 podemos observar el formato que tiene originalmente abierto con el Excel.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	28	79	4	1	38	2	2019	6	12	3	V	3	V	3	V	3	V	3
2	28	79	4	6	48	2	2019	6	12	000.3	V	000.3	V	000.3	V	000.2	V	000.2
3	28	79	4	7	8	2	2019	6	12	4	V	1	V	1	V	1	V	1
4	28	79	4	8	8	2	2019	6	12	33	V	42	V	36	V	19	V	10
5	28	79	4	12	8	2	2019	6	12	39	V	44	V	37	V	20	V	12
6	28	79	8	1	38	2	2019	6	12	9	V	9	V	9	V	9	V	9
7	28	79	8	6	48	2	2019	6	12	000.1	V	000.1	V	000.1	V	000.1	V	000.1
8	28	79	8	7	8	2	2019	6	12	2	V	1	V	1	V	1	V	1
9	28	79	8	8	8	2	2019	6	12	36	V	31	V	10	V	9	V	10
10	28	79	8	9	47	2	2019	6	12	5	V	7	V	5	V	4	V	1

Figura 18: Datos en tiempo real de la calidad del aire

El significado de cada campo (aunque en el fichero no incluya la cabecera) es:

- Concatenando las columnas A, B y C: código de la estación
- Columna D: magnitud medida
- Columna G: año de la medida
- Columna H: mes de la medida
- Columna I: día de la medida
- Columna J: corresponde a H01 y es el valor de la magnitud en la primera hora del día
- Columna K: si es V, el valor del registro previo es correcto. En caso de ser N es inválido.
- Columna L: corresponde a H02 y es el valor de la magnitud en la segunda hora del día
- Columna M: si es V, el valor del registro previo es correcto. En caso de ser N es inválido.

Para procesarlos y dejarlos en un formato amigable para su posterior uso empleamos la función definida en el fragmento de código 10. En esta función se invoca a la que se ha visto previamente para que descargue el *dataset* y lo procesa creando uno nuevo formateado.

```
1 def tratar_dataset_tiempo_real():
2     """Tras invocar a la funcion get_tiempo_real() para la descarga del dataset
          con las medidas en tiempo real de las estaciones, lo procesa.
3     Genera un nuevo fichero unico con las medidas de todas las estaciones.
4
5     Devuelve un string con la url donde se encuentra almacenado el fichero
          salida
```

```
6
7     """
8
9     ...
```

Fragmento de código 10: Función tratar_dataset_tiempo_real de datasets.py

En la figura 19 podemos ver el ejemplo del *dataset* ya formateado.

	A	B	C	D	E
1	ESTACION	MAGNITUD	FECHA	HORA	VALOR
2	28079004	so2	12/06/2019	18:00	1
3	28079004	co	12/06/2019	18:00	000.3
4	28079004	no2	12/06/2019	18:00	21
5	28079008	so2	12/06/2019	18:00	9
6	28079008	co	12/06/2019	18:00	000.2
7	28079008	no2	12/06/2019	18:00	46
8	28079008	pm10	12/06/2019	18:00	5
9	28079008	pm2_5	12/06/2019	18:00	14
10	28079008	o3	12/06/2019	18:00	86
11	28079011	no2	12/06/2019	18:00	41
12	28079016	co	12/06/2019	18:00	000.2
13	28079016	no2	12/06/2019	18:00	23

Figura 19: Datos en tiempo real de la calidad del aire procesados

***Dataset* Datos climatológicos en tiempo real**

De los objetos del fichero JSON descargado utilizaremos los atributos:

- "viento": dentro del objeto viento hacemos la media de las velocidades ya que viene un valor por cada tramo horario.
- "temperatura": dentro del objeto temperatura se hará la media de los valores que tomará esta magnitud cada hora.

El valor de las precipitaciones, como comentamos en el capítulo 5.1, de captura de datos, se obtiene directamente por *web scrapping*.

Los datos los descargamos mediante la función del fragmento de código 11.

```
def get_climatologia_tiempo_real():
    """Descarga el dataset con la informacion de la climatologia en el dia. Lo
       guarda en disco.
    El valor de las precipitaciones en mm, se obtiene de la web www.eltiempo.es
       ya que el dataset anterior cuenta con el porcentaje de probabilidad de
       lluvia, pero no con el valor en mm

    Devuelve el valor de las precipitaciones obtenidas por web scrapping

    Excepciones:
    ValueError -- Si la funcion descargarArchivosUrl propaga algun error.

    """

    api_key = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJsdXptYXJpYS5tdHRAZ21haWwuY
    29tIiwianRpIjoiM2VhNDVkZTgtYjFhZS00Y2M1LTllY2UtYjZmZDgwZGI4OGEyIiwiaX
    NzIjoiQUVNRVQiLCJpYXQiOjE1NjAzMzg5NjYsInVzZXJJZCI6IjNlYTQ1ZGU4LWIxYWU
    tNGNjNS05ZWNlLWI2ZmQ4MGRiODhhMiIsInJvbGUiOiIifQ.TxnmkwDmYfXhQFPjVE_nE
    uEd5fcWugbenooNuypGX3w"
    url = "https://opendata.aemet.es/opendata/api/prediccion/especifica/
    municipio/diaria/28079"

    ...
```

Fragmento de código 11: Función get_climatologia_tiempo_real de datasets.py

Una vez tenemos los datos en local. Se procesan a través de la función del fragmento de código 12 dejándolos ya formateados para su posterior uso.

```
def tratar_dataset_climatologia_tiempo_real():
    """Tras invocar a la funcion get_climatologia_tiempo_real() para la
       descarga del dataset con las medidas en tiempo real de la climatologia,
       lo procesa.

    Devuelve tres valores, la temperatura, el viento y las precipitaciones.
```

```
5
6     """
7
8     ...
```

Fragmento de código 12: Función tratar_dataset_climatologia_tiempo_real de datasets.py

5.4. Modelos de predicción

Para generar los modelos de predicción vamos a servirnos del área de aprendizaje supervisado de la inteligencia artificial. En concreto vamos a implementar árboles de decisión (mediante clasificación) y modelos de redes neuronales clasificadoras. Al no medir las estaciones las mismas magnitudes, los atributos de entrada varían, por lo que se creará un árbol y una red neuronal por estación.

Para el desarrollo del código y la implementación de los modelos, utilizaremos la librería de Python scikit-learn (https://scikit-learn.org/stable/install.html).

5.4.1. Árboles de decisión

Los árboles de decisión es una de las técnicas más usadas para el aprendizaje supervisado y aunque es bastante eficaz contra los datos ruidosos, en nuestro programa los datos que usa como valores de atributos en la entrada están limpios sin valores nulos. Aunque ese hecho no exime de que algún dato del *dataset* original pueda ser incorrecto. La función toma la forma de un árbol y se suele interpretar como condicionantes anidados que se pueden traducir de forma sencilla a reglas.

El funcionamiento del árbol de decisión consiste en que cuando tenemos una instancia sin clasificar, los diferentes valores de los atributos se van comparando con las ramas del árbol (es decir, se va comprobando las diferentes condiciones y de esta forma se recorre el árbol hacia abajo), finalmente se llegará a un nodo hoja, siendo este la clase a la que pertenece la instancia.

En nuestro caso los atributos de entrada serán los valores de las magnitudes y del clima (temperatura, velocidad del viento y precipitaciones) y las posibles clases de salida serán los

niveles de predicción (0,1,2 ó 3)

Consideramos que es adecuado aplicar este modelo porque la función objetivo tiene valores de salida discretos (los niveles de predicción), son sencillos de comprender y el mapeo a reglas es fácil, además pueden trabajar con datos multidimensionales y no requieren establecer parámetros a priori.

Y aunque sería más recomendable quizás, que los atributos de entrada fueran categóricos en vez de numéricos, no es un problema insalvable ya que los árboles también cuentan con algoritmos para trabajar con este tipo de datos.

En el trabajo se ha implementado un árbol por cada estación de calidad del aire porque no todas tienen la misma cantidad de atributos de entrada al no medir todas las mismas magnitudes.

El árbol se genera mediante la función del fragmento de código 13 que recibe como parámetro el código de la estación, la profundidad que queremos que tenga como máximo el árbol (si no se indica por defecto es 5) y si deseamos que se genere un archivo con las estadísticas (por defecto se exporta).

```
1  def generar_arbol (codigo, profundidad=5, estadisticas=True):
2      """Genera un arbol de decision (clasificador)
3
4      Devuelve el arbol generado y el porcentaje de acierto al validar el modelo
           con el conjunto de entrenamiento.
5
6      Parametros:
7      codigo -- codigo de la estacion de calidad del aire
8      profundidad -- profundidad maxima del arbol. Por defecto 5
9      estadisticas -- True si se quiere exportar un fichero con estadisticas.
           False si no se quiere generar. Por defecto True
10
11     """
12
13     ...
```

Fragmento de código 13: Función generar_arbol de trees.py

Para validar el modelo se utiliza la técnica de validación cruzada, que consiste en dividir el conjunto de datos disponible en dos, uno se usará para entrenar el modelo mientras que el otro se empleará para validarlo. La partición la realizamos mediante la función *train_test_split* (ver fragmento de código 14). Para la división de los datos usamos los siguientes parámetros:

- l_atributos: lista con los diferentes conjuntos de atributos de entrada (valores).
- l_etiquetas: lista con los valores que toma la clase para cada registro (n_prediccion).
- test_size: porcentaje de datos del conjunto que queremos que se usen para validar. En nuestro caso utilizaremos el 75 % de los datos para entrenar y el 25 % para validar.
- random_state: en el caso de no especificar este parámetro se obtendrá un resultado en la división de los datos diferente cada vez que se ejecute la función. En cambio, si se indica el parámetro con un valor concreto se puede garantizar que la salida de todas las ejecuciones será la misma, es decir, la división que realiza en los datos y los subconjuntos resultantes serán iguales. Para las pruebas se va a dejar fijo, pero sería recomendable que el día que pasara a producción el software no se especificara para que las divisiones fueran realmente aleatorias.

```
1  #Separamos los datos que vamos a utilizar para entrenar y para validar.
2  #datos de validacion (0.25) / datos de entrenamiento (0.75)
3  X_train, X_test, y_train, y_test = train_test_split(l_atributos,l_etiquetas,
4      test_size = 0.25,random_state = 0)
```

Fragmento de código 14: Función train_test_split de la librería sklearn

Una vez tenemos los subconjuntos de datos creados les ajustamos la escala, es decir, los vamos a normalizar. Para estandarizar los datos se transformarán de tal forma que su distribución tendrá un valor de la media 0 y desviación estándar de 1. Por lo tanto, para llevar a cabo dicha transformación en la librería sklearn contamos con la función *fit()* que calcula los parámetros necesarios y los guarda con un objeto interno, sólo es necesario llamarla una vez. Después llamaríamos al método *transform()* para aplicar la transformación. En nuestro caso hemos usado la función *fit_transform()* que agrupa estos dos métodos en uno. (Ver fragmento de código 15)

```
#Normalizacion de los datos
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Fragmento de código 15: Normalización de los datos con la librería sklearn

Creamos el árbol de decisión con el método *DecisionTreeClassifier()* ya que es un árbol de clasificación y no de regresión. Como parámetros utilizamos:

- max_depth: profundidad máxima que queremos que tenga el árbol.
- criterion: utilizaremos 'entropy' ya que está más enfocado para valores discretos. Si tuvieramos valores continuos podríamos usar 'gini'.
- random_state: semilla usada para generar los números aleatorios. Si la fijamos siempre será igual.
- splitter: no especificamos este parámetro porque queremos dejarlo por defecto, la mejor. Es la estrategia usada para escoger la división en cada nodo. Otra alternativa podría ser al azar.

Es recomendable fijar la profundidad máxima ya que si no se define ese parámetro tendríamos el problema del sobreajuste (100 % de éxito en la validación). Y probablemente haría que no fuera un buen clasificador.

La creación del árbol la podemos ver en el fragmento de código 16.

```
#Creamos el arbol con la profundidad indicada
arbol = tree.DecisionTreeClassifier(max_depth=profundidad,
    criterion = 'entropy', random_state = 0)
```

Fragmento de código 16: Creación de un árbol de decisión clasificador con la librería sklearn

Una vez tenemos el árbol hay que entrenarlo, para ello usamos el método *fit()* como podemos observar en el fragmento de código 17. Como parámetros le pasaremos los subconjuntos que creamos previamente para el entrenamiento.

```
#Entrenamos el arbol
arbol.fit(X_train, y_train)
```

Fragmento de código 17: Entrenamiento del árbol con la librería sklearn

En este momento, vamos a exportar el árbol de dos formas diferentes, una como un archivo DOT por si se quisiera graficar posteriormente y otra como un TXT en la que veremos el árbol mapeado en reglas. Los métodos empleados para ambos se encuentran dentro de la librería sklearn. Los podemos ver en el fragmento de código 18.

```
#Exportar el arbol en archivo de texto llamado export_tree_text_<codigo
    estacion>.txt. Como paramteros pasaremos el arbol y los nombres de los
    atributos.
r = export_text(arbol, feature_names=cabeceras[:-1])
f=open(path + "export_tree_text_" + str(codigo) + ".txt","w")
f.write(r)
f.close()

#Exportar datos del arbol en un fichero .dot llamado export_tree_<codigo
    estacion>.dot. Como parametros relevantes pasamos el arbol, la ubicacion
    del archivo donde se quiere guardar, el nombre de las clases y el nombre
    de los atributos.
export_graphviz(arbol,out_file=path + 'export_tree_' + str(codigo) + '.dot'
    ,class_names=clases, feature_names=cabeceras[:-1],
    impurity=False,filled=True)
```

Fragmento de código 18: Exportar el árbol con la librería sklearn

En lo que a los resultados se refiere, es el momento de crear la matriz de confusión, sacar reportes y ver el porcentaje de acierto. La librería sklearn cuenta con funciones y métodos para todo ello. En el fragmento de código 19 se muestra cómo hacerlo.

```
# Predicion de los resultados del bloque test
y_pred = arbol.predict(X_test)
#Matriz de confusion
cm = confusion_matrix(y_test, y_pred)
#Reportes
report = classification_report(y_test, y_pred)
#Score con los datos de vaidacion
arbol.score(X_test, y_test)
#Score con los datos de entrenamiento
arbol.score(X_train, y_train)
```

Fragmento de código 19: Estadísticas con la librería sklearn

En algunos casos al entrenar el árbol, Python nos lanza por consola un warning como el que podemos ver en la figura 20 (en este caso sólo aparece para una de las estaciones).

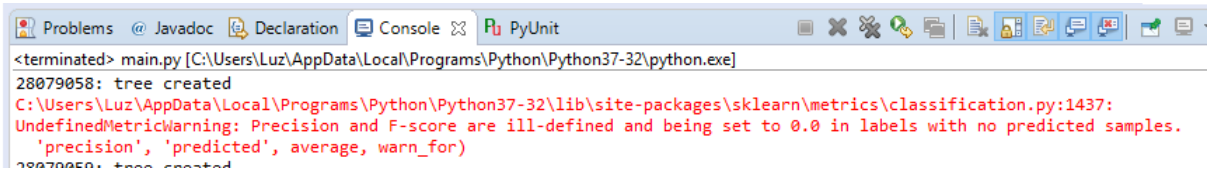


Figura 20: *Warning* emitido por python en árboles

Es debido a que algunas etiquetas (clases) de y_test no aparecen en y_pred. Por lo que no hay un puntaje f para calcular para esta etiqueta y, por lo tanto, el puntaje f para este caso se considera 0.0. Como se ha solicitado un promedio de la puntuación, se ha incluido una puntuación de 0 en el cálculo, y es por eso que scikit-learn muestra la advertencia. En caso de que se quisiera evitar que este *warning* se muestre, se podría hacer de dos maneras, bien inhibiendolos con *warnings.filterwarnings('ignore')*, o sin ser tan drásticos, a la hora de pasar los parámetros para calcular las métricas indicar que únicamente estamos interesados en los puntajes de las etiquetas que se predijeron, para ello especificaríamos las etiquetas que nos aplican *labels=np.unique(y_pred)* (que son etiquetas que se predijeron al menos una vez).

A continuación se muestra un ejemplo de un árbol con profundidad 6 creado para la estación 28079024 y las estadísticas obtenidas.

En la figura 21 podemos observar el árbol generado mapeado en reglas.

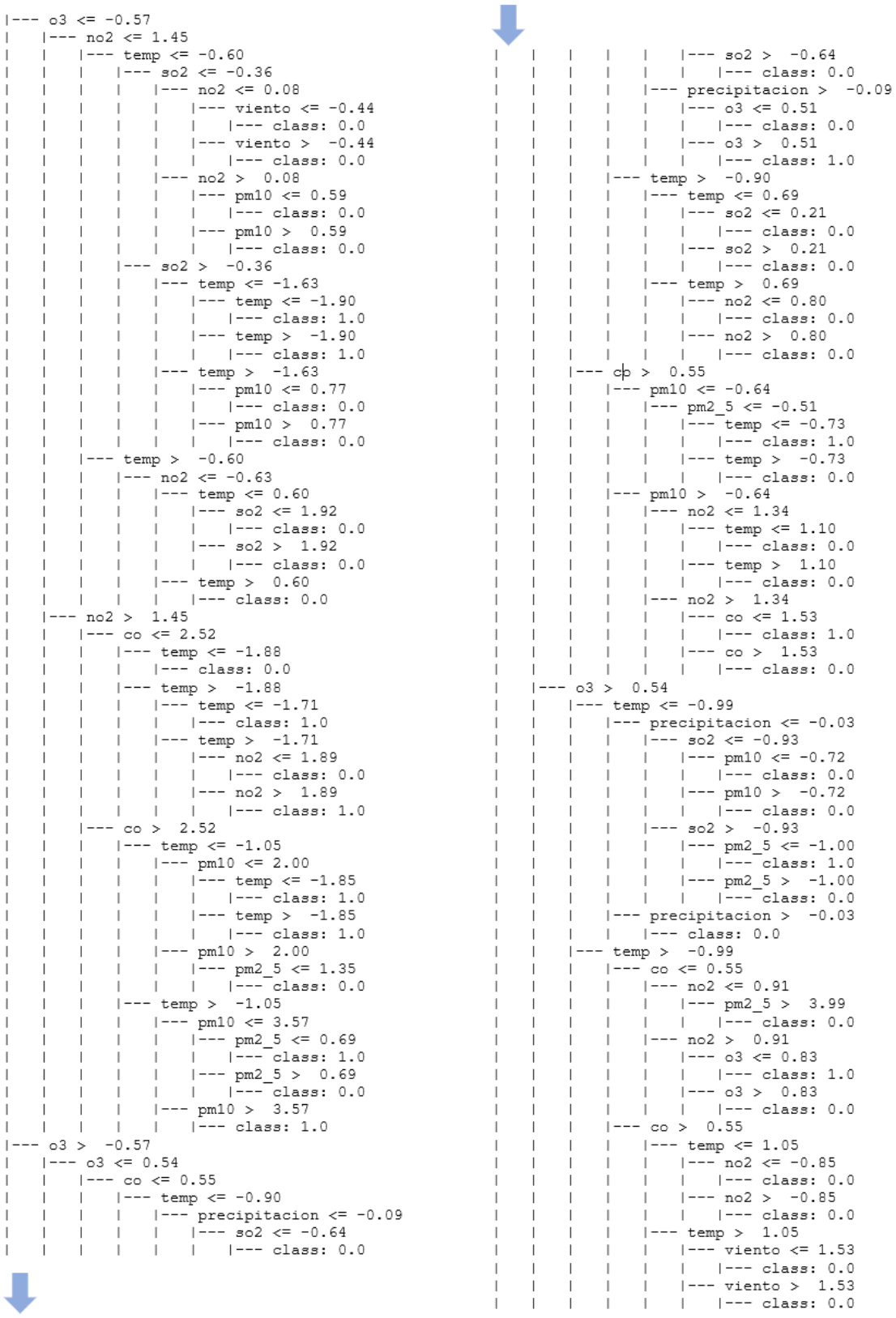


Figura 21: Árbol con profundidad 6 generado para 28079024

Dentro de las estadísticas que se generan, una de las más relevantes es la matriz de confusión que nos permite visualizar el rendimiento del modelo predictivo, es decir, es una tabla que describe el rendimiento de un modelo supervisado con los datos de prueba, donde se desconocen los verdaderos valores pudiendo detectar de forma sencilla dónde se está confundiendo el sistema. Está formada por 4 valores para una matriz de 2x2:

- *True Positives* (TP): cuando la clase real de los datos era 1 (verdadero) y la predicha es también 1 (verdadero).
- *True Negatives* (TN): cuando la clase real de los datos era 0 (falso) y la predicción también es 0 (falso).
- *False Positives* (FP): cuando la clase real de los datos era 0 (falso) y la predicha es 1 (verdadero).
- *False Negatives* (FN): cuando la clase real de los datos era 1 (verdadero) y la predicha es 0 (falso).

En la figura 22 podemos ver de forma gráfica cómo estarían situados los valores anteriores dentro de la matriz.

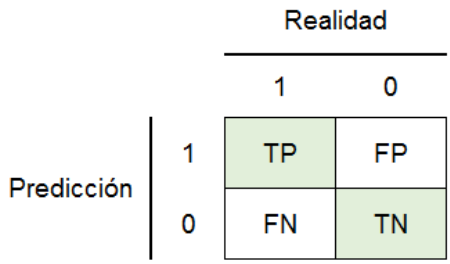


Figura 22: Matriz de confusión

Partiendo de la matriz de confusión obtenemos el resto de métricas:

- Precisión: es el porcentaje total de elementos clasificados correctamente. Es la medida que mejor refleja la calidad siempre que el conjunto de datos esté equilibrado, toma un valor entre 0 y 1 y cuanto más alto sea, mejor.

$$precision = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

Tiene una gran desventaja y es que si los datos no están balanceados no es la medida más adecuada. Ya que por ejemplo, si contamos con el 90 % de los datos de una clase

y el 10 % de la otra, la precisión puede tomar el valor de 0.9, que sería bastante bueno, pero puede que no haya clasificado bien ninguna instancia del conjunto del 10 %.

- Recall: es la sensibilidad o tasa de *True Positive*, es decir, es el número de elementos identificados correctamente como positivos del total de positivos verdaderos. Nos da la información respecto a los falsos negativos, es decir, cuantos fallaron.

$$recall = \frac{TP}{TP + FN} \tag{2}$$

- F1-score: es la medida de precisión empleada en la determinación de un valor único ponderado de la precisión y la exhaustividad, es decir, es una media armónica. Penaliza más los errores al clasificar ejemplos positivos (FN) que los errores al clasificar ejemplos negativos (FP).

$$f1 - score = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{3}$$

  Al ser un valor único nos permite comparar dos modelos de predicción entre sí.

A continuación, en la figura 23 vemos las estadísticas que se han generado para la estación 28079024.

```
************************ CLASS ************************
0.0
1.0
********************* MAX DEPTH ************************
6
***************** FEATURE IMPORTANCES ******************
so2: 0.054290828595643756
co: 0.0868862395562579
no2: 0.12450288195637957
pm10: 0.03783450126579918
pm2_5: 0.02964591481068919
o3: 0.4537980803066583
temp: 0.17759409952426758
viento: 0.012133954053463831
precipitacion: 0.023313499930840687
************************ SCORE ************************
With Test data: 0.8604954367666232
With Training data: 0.8775820830615351
****************** CONFUSION MATRIX ********************
        0       1
----------------------
0 |     1269    52
1 |     162     51
************************ REPORT ************************
              precision    recall  f1-score   support

         0.0       0.89      0.96      0.92      1321
         1.0       0.50      0.24      0.32       213

    accuracy                           0.86      1534
   macro avg       0.69      0.60      0.62      1534
weighted avg       0.83      0.86      0.84      1534
```

Figura 23: Estadísticas del árbol con profundidad 6 generado para 28079024

Si analizamos el contenido podemos ver que hay dos clases, la 0 y la 1. Aunque en realidad hay 4 clases, los datos de entrenamiento de esta *dataset* sólo tienen 2, la 0 y la 1, por lo que no puede conocer las otras dos. Indica que la profundidad máxima del árbol es 6. En el siguiente apartado, *feature importances*, podemos observar la importancia que tiene cada atributo en este modelo de árbol. En la figura 24 podemos ver estos mismos datos de forma gráfica, ordenados por el porcentaje de importancia. A simple vista se ve cómo la magnitud O_3 tiene prácticamente la misma importancia que el resto juntas.

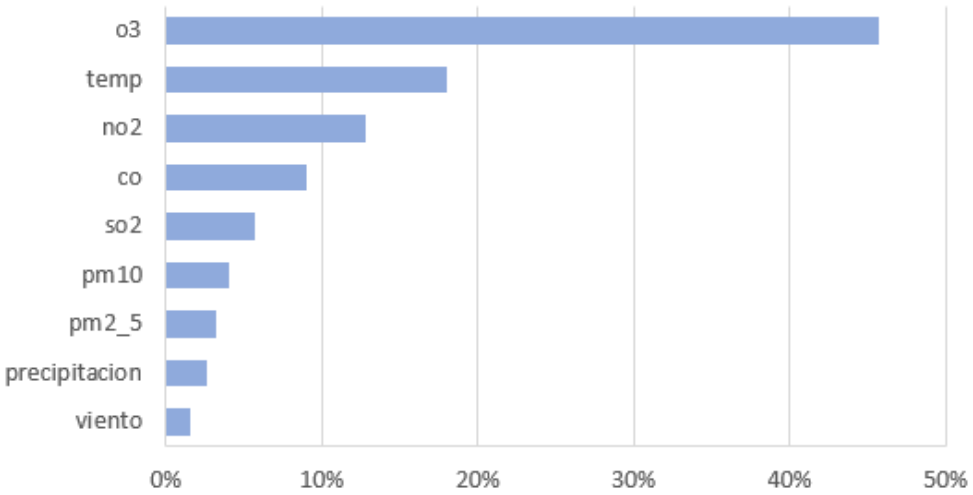


Figura 24: Importancia de cada atributo del árbol de 28079024

Respecto a los porcentajes de acierto, vemos que con el conjunto de datos de validación ha conseguido clasificar correctamente aproximadamente el 86 % mientras que con los datos de entrenamiento consiguió clasificar prácticamente un 88 %.

La matriz de confusión es de 2x2 porque en este caso sólo contamos con dos clases. En ella podemos ver que de un total de 1534 instancias, 1269 que eran de la clase 0 se han clasificado correctamente en la clase 0, 51 que pertenecían a las clase 1 se han clasificado con éxito en la clase 1, mientras que 162 instancias pertenecientes a la clase 1 se han clasificado erróneamente en la 0 y 52 pertenecientes a la 0 se han clasificado equivocadamente en la clase 1.

En el reporte vemos de forma más clara y concisa como el modelo tiene un 89 % de precisión para la clase 0 que está muy bien, pero en cambio para la clase 1 sólo un 50 %. Probablemente sea debido a la cantidad de instancias con las que cuenta de una clase y de otra. Ya que de la clase 0 tiene 1321 registros mientras que de la clase 1 sólo 213. La columna *f1-score* nos indica el porcentaje de predicciones positivas que fueron correctas. Estos valores son normalmente más bajos que los de precisión porque incorporan precisión y recuperación en sus cálculos. Normalmente este valor se emplea para comparar modelos de clasificadores, no la precisión global.

5.4.2. Redes neuronales

Las redes neuronales se han convertido en los algoritmos de inteligencia artificial más populares, pero como modelo computacional existen desde mediados del siglo pasado, pero no ha sido hasta hace algunos años con la mejora de la técnica y la tecnología que se han empezado a usar masivamente.

Se pueden emplear para múltiples usos como reconocimientos de caracteres, de imágenes, de voz, predicciones, traducción de idiomas, prevención de fraude, generación de texto, conducción autónoma, análisis genético, pronóstico de enfermedades, etc. Como podemos intuir se trata de una familia de algoritmos muy potentes con los que podemos modelar comportamientos inteligentes.

Al ser una estructura evolucionada y tener comportamientos avanzados su complejidad emerge de la interacción de muchas partes más simples trabajando conjuntamente, en el caso de la red neuronal, a cada una estas partes la denominaríamos neurona.

La neurona es la unidad básica de procesamiento dentro de una red neuronal. Es similar a una neurona biológica por lo que tienen conexiones de entrada a través de los que reciben estímulos externos (valores de entrada). Con estos valores, la neurona realizará un cálculo interno y generará un valor de salida. En el cálculo, la neurona utiliza todos los valores de entrada para realizar una suma ponderada de ellos, la ponderación de cada una de las entradas viene dada por el peso que se le asigna a cada una de las conexiones de entrada, es decir, cada conexión que llega a la neurona tendrá asociado un valor que servirá para definir con qué intensidad cada variable de entrada afecta a la neurona. Estos pesos son lo parámetros del modelo y serán los valores que podremos ajustar para modificar el funcionamiento de la red neuronal. También existe otra variable de entrada independiente denominada sesgo que por defecto valdrá 1, es modificable.

Las neuronas se pueden combinar entre sí, en forma de capas (las neuronas estarán situadas en una columna vertical) donde todas las neuronas recibirán la misma información de entrada procedente de la capa anterior. Los cálculos que se obtengan los pasarán por una función de activación y llegarán a la siguiente capa, que dependiendo del diseño de la red, según el resultado, activará las siguientes neuronas o no. A la primera capa donde están las variables de entrada se la denomina capa de entrada, y a la última, capa de salida, mientras que a las capas intermedias se las denomina capas ocultas. Cuando las neuronas están si-

tuadas de manera secuencial, una recibe la información procesada de la otra, de esta forma la red puede aprender conocimiento jerarquizado.

Las redes neuronales tienen muchas bondades al estar basadas en la estructura del sistema nervioso: aprendizaje, ya que se le pueden proporcionar datos como entrada a la vez que se le indica cuál debe ser su salida; autoorganización, la red neuronal crea su propia representación de la información en su interior, por lo que descarga al usuario de esta parte; tolerancia a fallos, flexibilidad en los cambios no importantes; respuesta en tiempo real al ser una estructura paralela; etc. Pero también cuenta con desventajas como por ejemplo la complejidad en el aprendizaje para gran cantidad de datos, es decir, cuantas más cosas se quiere que aprenda la red, más complicado es enseñarle; el tiempo de aprendizaje es elevado; la red no puede ser fácilmente interpretada por el usuario como por ejemplo un árbol; etc.

En nuestro caso usaremos las redes neuronales como redes de aprendizaje supervisado, donde los datos de entrada que serán los valores de las magnitudes y del clima (temperatura, velocidad de viento y precipitaciones) y las posibles salidas serán los niveles de predicción (0, 1, 2 ó 3).

En el trabajo, al igual que en el caso de los árboles, se ha implementado una red neuronal por cada estación de calidad del aire porque no todas tienen la misma cantidad de atributos de entrada al no ser uniformes las magnitudes que se miden.

La red neuronal se genera mediante la función del fragmento de código 20 que recibe como parámetros de entrada el código de la estación y si queremos que se genere un archivo con las estadísticas (por defecto se exporta).

```
1 def generar_red_neuronal (codigo, estadisticas=True):
2     """Genera una red neuronal (clasificadora)
3
4     Devuelve la red neuronal generada y el porcentaje de acierto al validar el
          modelo con el conjunto de entrenamiento.
5
6     Parametros:
7     codigo -- codigo de la estacion de calidad del aire
8     estadisticas -- True si se quiere exportar un fichero con estadisticas.
          False si no se quiere generar. Por defecto True
9
```

```
10
11    """
12
13    ...
```

Fragmento de código 20: Función generar_red_neuronal de neural_networks.py

Para validar el modelo se utiliza la técnica de validación cruzada, al igual que en el módelo de los árboles. Se realiza con la misma función y de igual manera, empleando el 25 % de los datos para validar y el 75 % para el conjunto de datos de entrenamiento.

Para crear la red neuronal utilizamos el método *MLPClassifier()*. Se implementa un algoritmo de perceptrón multicapa que se entrena con *backpropagation* (algoritmo de aprendizaje que consigue que una red neuronal autoajuste sus parámetros para así aprender una representación interna de la información que está procesando). Como parámetros utilizamos:

- max_iter: número máximo de iteraciones, por defecto 200.
- hidden_layer_sizes: número de neuronas en la capa oculta, por defecto 100.

La creación de la red neuronal la podemos ver en el fragmento de código 21.

```
1    #Creamos la red neuronal
2    red = MLPClassifier(max_iter=100000, hidden_layer_sizes=(50,25))
```

Fragmento de código 21: Función creación de la _red_neuronal con la librería sklearn

Una vez tenemos la red neuronal hay que entrenarla. Para ello usamos el método *fit()* como podemos observar en el fragmento de código 22.

```
1    #Entrenamos la red neuronal
2    red.fit(X_train, y_train)
```

Fragmento de código 22: Entrenamiento de la _red_neuronal con la librería sklearn

Respecto a los resultados, en este punto, creamos la matriz de confusión, se sacan los reportes y se puede ver el porcentaje de acierto. Al igual que con los árboles, la librería sklearn también cuenta con funciones y métodos para todo ello respecto a las redes neuronales. En el fragmento de código 23 se muestra cómo hacerlo.

```
#Predicion de los resultados del bloque test
y_pred = red.predict(X_test)
#Matriz de confusion
cm = confusion_matrix(y_test, y_pred)
#Reportes
report = classification_report(y_test, y_pred)
#Score con los datos de validacion
red.score(X_test, y_test)
#Score con los datos de entrenamiento
red.score(X_train, y_train)
```

Fragmento de código 23: Estadísticas de la _red_neuronal con la librería sklearn

Al entrenar las diferentes redes neuronales para las estaciones en dos de ellas, nos aparece el *warning* de que alguna de las etiquetas el puntaje f es 0.0 y se está teniendo en cuenta para los cálculos de puntuación. El caso es similar al que nos ocurría en el árbol. Podemos ver los *warning* en la Figura 25.

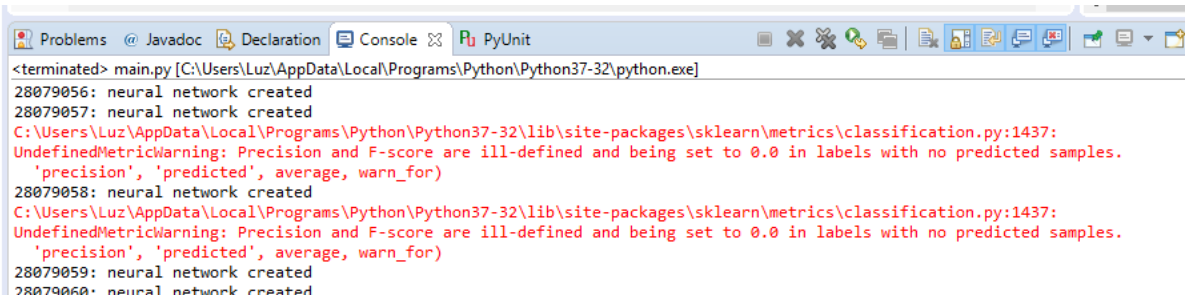


Figura 25: *Warning* emitido por python en redes neuronales

A continuación se muestra en la figura 26, para la estación 28079024, las estadísticas obtenidas.

```
************************ CLASS ***************************
0.0
1.0
********************** FEATURES  ***********************
so2
co
no2
pm10
pm2_5
o3
temp
viento
precipitacion
************************ SCORE ***************************
With Test data: 0.8676662320730117
With Training data: 0.8688845401174168
****************** CONFUSION MATRIX ********************
        0       1
--------------------
0 |     1317    4
1 |     199     14
************************ REPORT ************************
              precision    recall  f1-score   support

         0.0       0.87      1.00      0.93      1321
         1.0       0.78      0.07      0.12       213

    accuracy                           0.87      1534
   macro avg       0.82      0.53      0.52      1534
weighted avg       0.86      0.87      0.82      1534
```

Figura 26: Estadísticas de la red neuronal generada para 28079024

Si analizamos el contenido podemos ver que hay dos clases, la 0 y la 1. Aunque en realidad hay 4 clases, los datos de entrenamiento de este *dataset* sólo tienen 2, la 0 y la 1, por lo que no se pueden predecir las otras dos al no conocerlas. En el siguiente apartado, *features*, observamos los atributos de entrada usados en esta red.

Respecto a los porcentajes de acierto, vemos que con el conjunto de datos de validación ha conseguido clasificar correctamente aproximadamente el 86.7 %, prácticamente el mismo porcentaje que obtuvo con el conjunto de entrenamiento, 86.8 %.

La matriz de confusión es de 2x2 porque en este caso sólo contamos con dos clases.

En ella podemos ver que de un total de 1534 instancias, 1317 que eran de la clase 0 se han clasificado correctamente en la clase 0, 14 que pertenecían a la clase 1 se han clasificado con éxito en la clase 1, mientras que 199 instancias pertenecientes a las clase 1 se han clasificado erróneamente en la 0 y 4 pertenecientes a las 0 se han clasificado equivocadamente en la clase 1.

En el reporte vemos de forma más clara y concisa como el modelo tiene un 87 % de precisión para la clase 0, siendo este bueno, pero en cambio para la clase 1 sólo acierta el 78 %. La columna *f1-score* nos indica el porcentaje de predicciones positivas que fueron correctas. Estos valores son normalmente más bajos que los de precisión porque incorporan precisión y recuperación en sus cálculos.

5.5. Funcionalidad del *software*

La funcionalidad principal del *software* desarrollado es recoger los datos climáticos diarios junto con las mediciones de las estaciones de calidad del aire y llevar a cabo la predicción a tres días vista, del nivel de contaminación que habrá.

En el archivo principal del programa, *main.py* inicialmente se llaman a las diferentes funciones que se han visto previamente para descargar los *datasets* históricos, procesarlos, generar los modelos (árbol y redes neuronales) para cada estación como podemos observar en el fragmento de código 24. Para la generación de los modelos de predicción sólo se han indicado a modo de ejemplo, como se crearía para dos estaciones.

```
1  if __name__ == '__main__':
2
3      #Descargamos el dataset con la estaciones de control
4      datasets.get_estaciones_control()
5      #Descargamos los historicos de las estaciones de control
6      datasets.get_historicos()
7      #Descargamos el historico de la climatologia
8      datasets.get_climatologia_historico()
9      #Procesamos el fichero de las estaciones de control y obtenemos el listado
           de estaciones
10     estaciones = datasets.tratar_dataset_estaciones_control()
```

```
11    #Procesamos el archivo historico de la climatologia
12    datasets.tratar_dataset_climatologia_historico()
13    #Procesamos los historicos de las estaciones de calidad del aire
14    #e incluimos las variables climatologicas procesadas previamente
15    datasets.tratar_dataset_historicos_training()
16    #Creamos los inputs para los clasificadores
17    datasets.crear_input_clasificadores_dataset ()
18
19    #Generamos los arboles para cada estacion
20    (arbol_28079017, score_arbol_28079017) = trees.generar_arbol (28079017, 6,
          True)
21    (arbol_28079018, score_arbol_28079018) = trees.generar_arbol (28079018, 6,
          True)
22
23    #Generamos las redes para cada estacion
24    (red_28079017, score_red_28079017) = neural_networks.generar_red_neuronal
          (28079017, True)
25    (red_28079018, score_red_28079018) = neural_networks.generar_red_neuronal
          (28079018, True)
```

Fragmento de código 24: Descargar y procesamiento de datasets. Creación de árboles y redes de main.py

En este punto tenemos dos sistemas de predicción por cada estación, por lo que escogeremos el mejor de los dos para cada una de ellas. Para ello, comparamos los resultados obtenidos al probarlos con el conjunto de datos de test (*score test*) y el que mayor puntaje haya obtenido será el modelo de predicción de esa estación. Como se puede ver en el fragmento de código 25, para la estación 28079017.

```
1    #Nos quedamos con el clasficador que tenga mejor score
2    if score_arbol_28079017 > score_red_28079017:
3        pred_28079017 = arbol_28079017
4    else:
5        pred_28079017 = red_28079017
```

Fragmento de código 25: Elección del modelo de predicción para cada estación de main.py

Finalmente cada hora, se obtienen los datos de climatología y los de las estaciones de calidad del aire, se procesan para que sean atributos de entrada del modelo de predicción y se obtiene el nivel de predicción. De este proceso también se obtienen los datos de entrada que permitirán mantener el *dashboard*:

- datos_magnitudes.csv: en el se van añadiendo cada actualización, por cada estación y magnitud, el valor, la fecha y la hora.
- predicciones.csv: se sobreescribe en cada actualización y en el se van anotando el nivel de predicción que se ha obtenido por cada estación.
- prediccion_total.csv: en este archivo se va añadiendo por cada actualización la fecha y el valor de la predicción máxima obtenida entre todas las estaciones.

Todo el código desarrollado que conforma el *software* se encuentra en el capítulo 9, Anexos u online en un repositorio de GitHub (https://github.com/luz-m/git-sistema-prediccion-contaminacion.git)

5.6. Entorno de visualización

Mediante Tableau se ha diseñado el *dashboard* de la figura 27. Consta de 5 gráficas, que a continuación se verán en detalle.

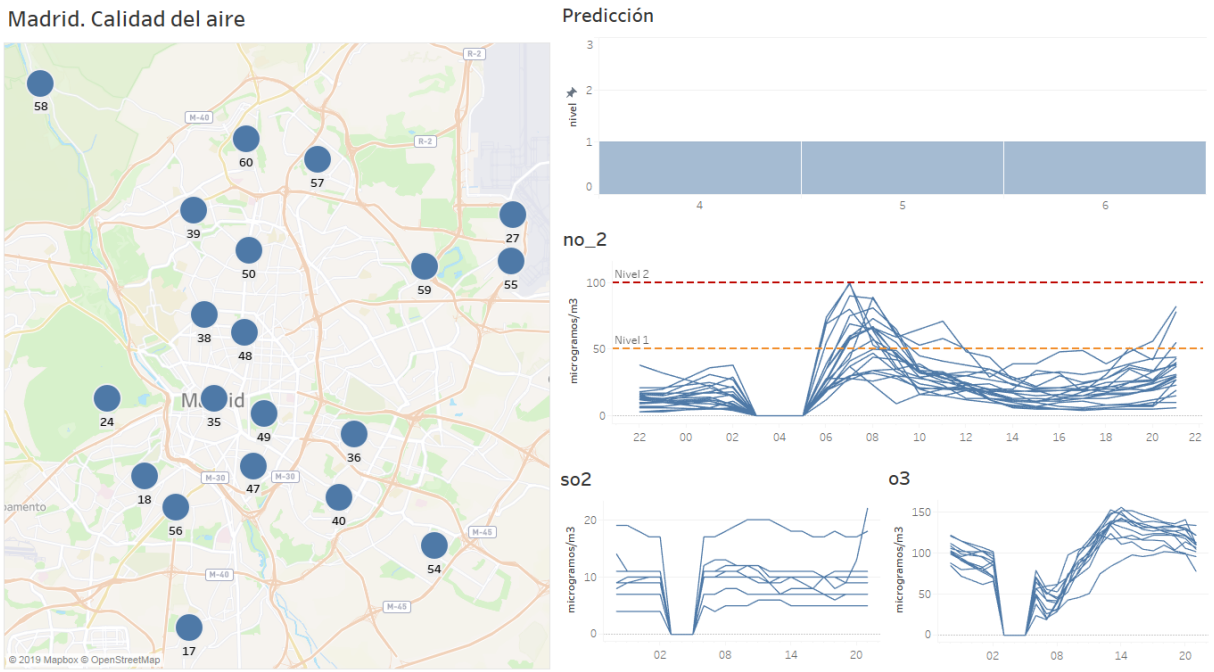
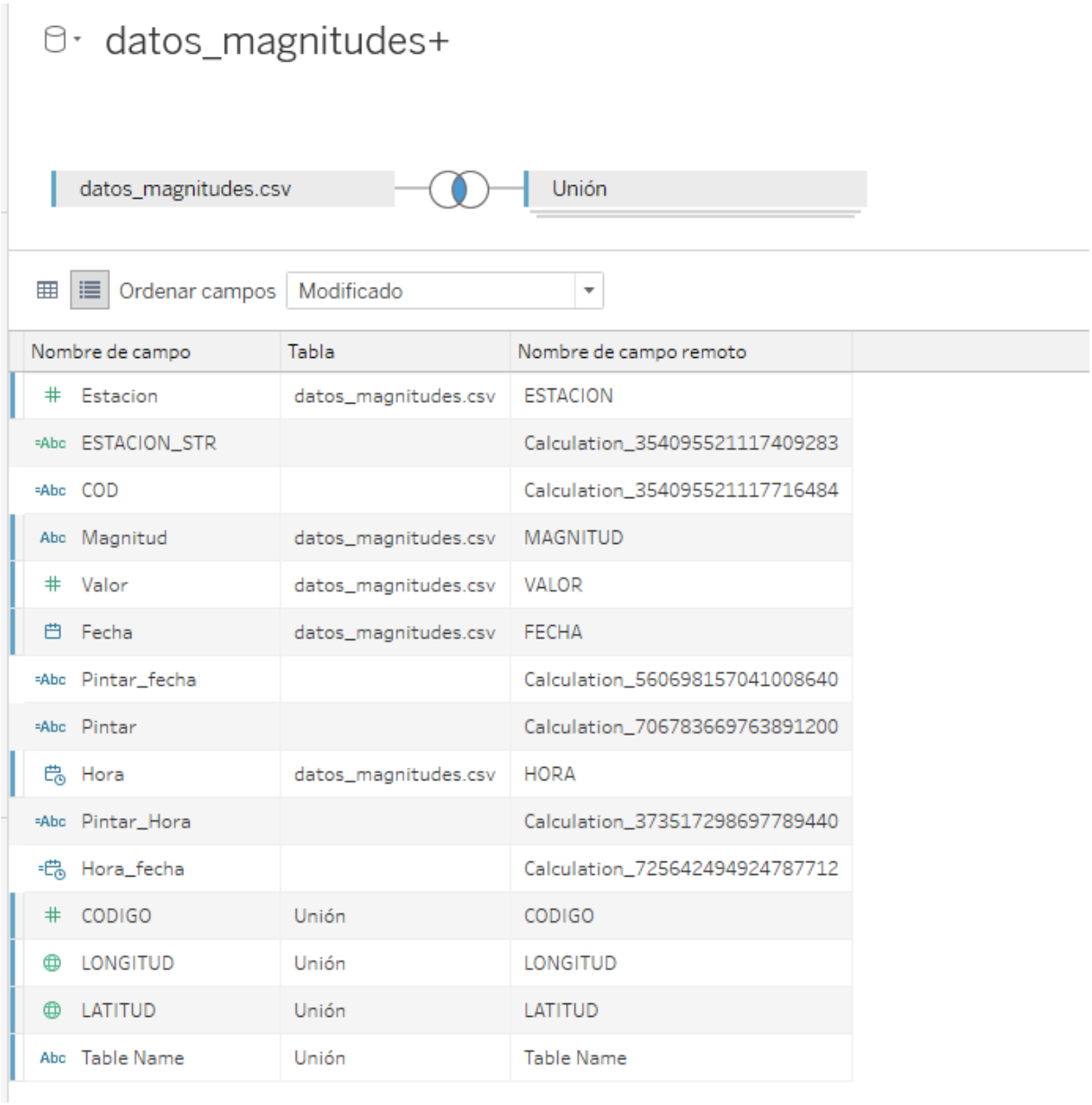


Figura 27: *Dashboard*

Para realizar el *dashboard* se han usado 3 fuentes de datos, 2 de ellas combinadas. Para el mapa y las gráficas de medidas de magnitudes, se han unido los archivos datos_magnitudes.csv (fichero donde se van registrando por cada estación y magnitud medida, su valor, fecha y hora, cada vez que se actualizan los datos en tiempo real) y estaciones.csv (archivo con un extracto de 212629-0-estaciones-control-aire.xls que contiene la estación, la longitud y la latitud únicamente). Además a esta fuente se le han añadido algunos campos calculados para decidir que valores se pintan en las gráficas y cuales no. En la figura 28 podemos ver los metadatos de la unión y en la figura 29 una vista previa de la fuente de datos.



Nombre de campo	Tabla	Nombre de campo remoto
Estacion	datos_magnitudes.csv	ESTACION
ESTACION_STR		Calculation_354095521117409283
COD		Calculation_354095521117716484
Magnitud	datos_magnitudes.csv	MAGNITUD
Valor	datos_magnitudes.csv	VALOR
Fecha	datos_magnitudes.csv	FECHA
Pintar_fecha		Calculation_560698157041008640
Pintar		Calculation_706783669763891200
Hora	datos_magnitudes.csv	HORA
Pintar_Hora		Calculation_373517298697789440
Hora_fecha		Calculation_725642494924787712
CODIGO	Unión	CODIGO
LONGITUD	Unión	LONGITUD
LATITUD	Unión	LATITUD
Table Name	Unión	Table Name

Figura 28: Fuente de datos 1: metadatos de la unión de datos_magnitudes.csv y estaciones.csv

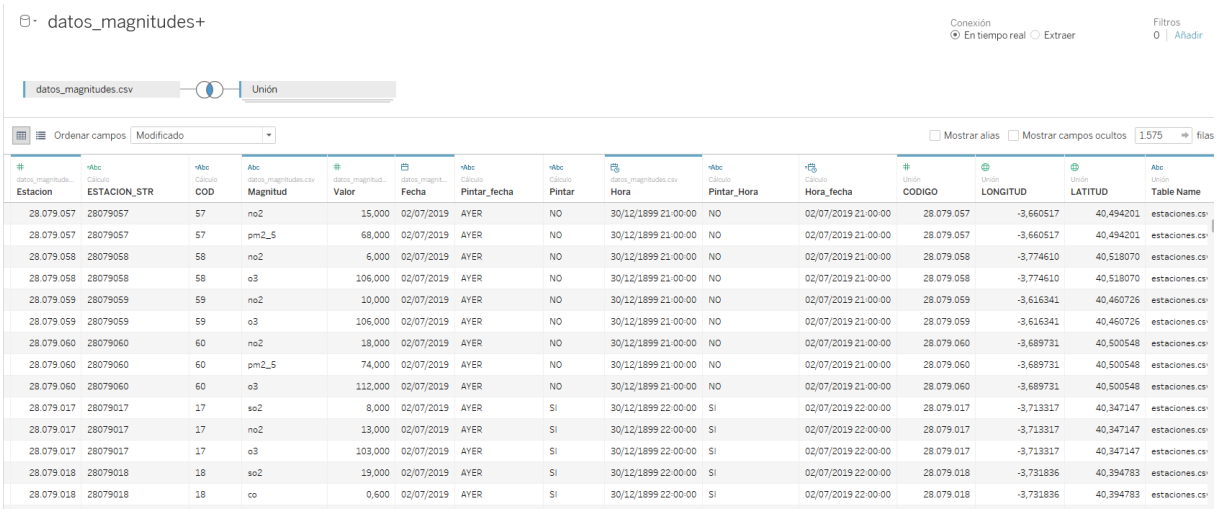


Figura 29: Fuente de datos 1: vista previa de la unión de datos_magnitudes.csv y estaciones.csv

La segunda fuente de datos únicamente proviene del fichero prediccion_total.csv donde cada vez que se actualizan los datos en tiempo real se añade una línea con la fecha y el valor máximo de predicción que se ha alcanzado entre todas las estaciones. En la figura 30 podemos observar los metadatos y en la figura 31 la vista previa de la fuente. Finalmente se añade un campo calculado donde se indica si ese registro entra dentro de los que se pueden graficar si procede o no.

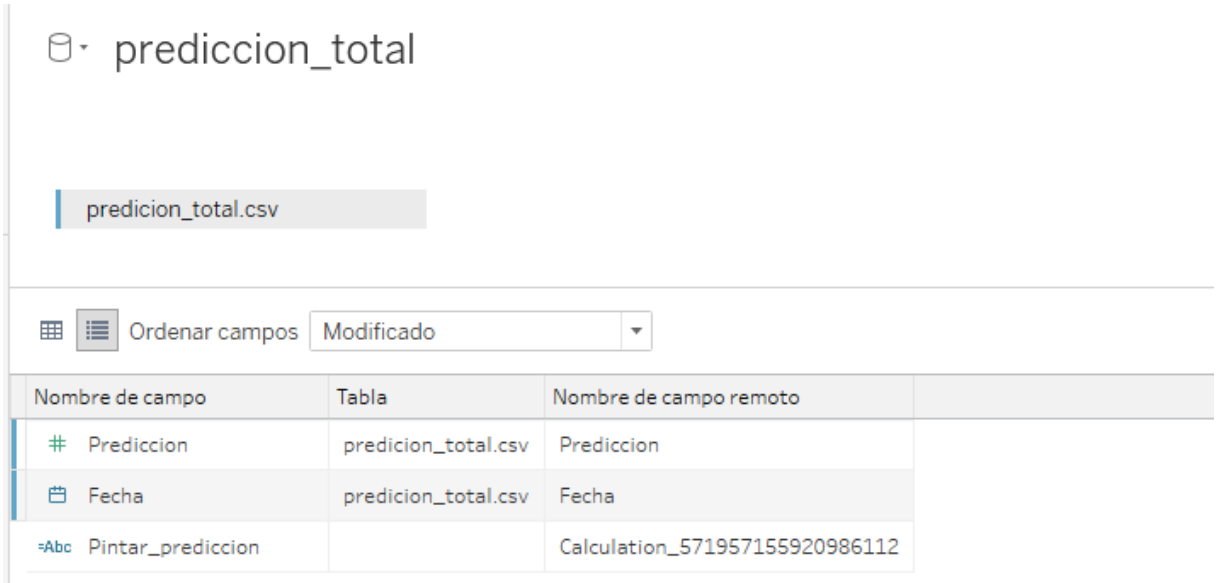


Figura 30: Fuente de datos 2: metadatos de prediccion_total.csv

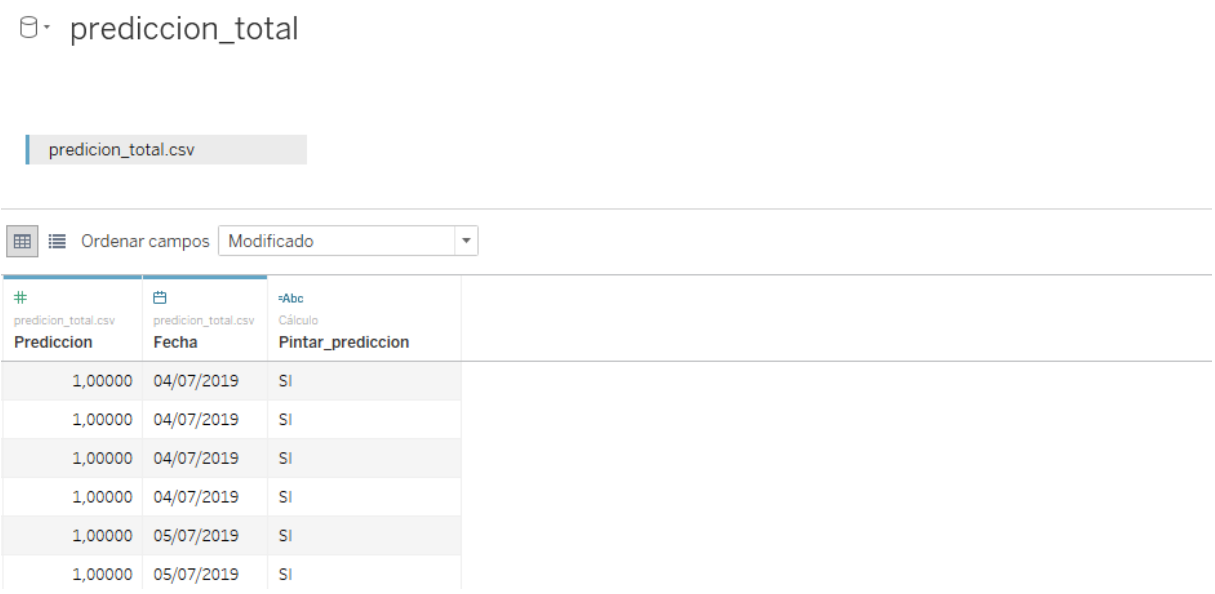


Figura 31: Fuente de datos 2: vista previa de prediccion_total.csv

En el caso de ambas fuentes, la conexión de datos es en tiempo real y no se ha extraído para que se pueda ir actualizando el *dashboard*.

Mapa de las estaciones: mapa en el que mediante la fuente datos_magnitudes+ se ubican las estaciones geográficamente. Para ello utilizamos la configuración de atributos y valores que podemos ver en la figura 32. Para mayor comodidad debajo de cada punto que simboliza una estación, no se ha indicado el código de la estación completo, si no los dos últimos dígitos únicamente (campo calculado COD).

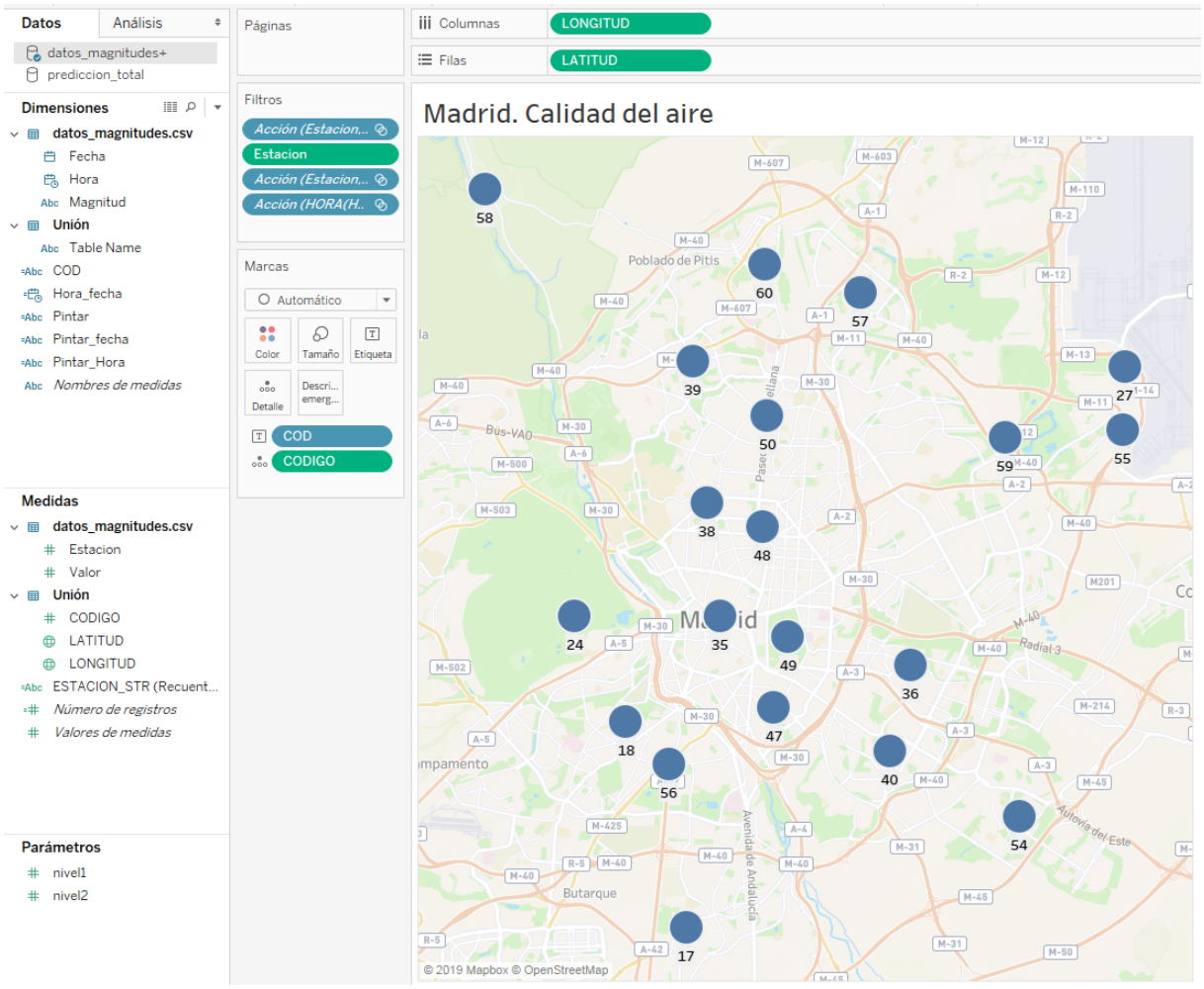


Figura 32: Configuración en Tableau gráfica mapa del *dashboard*

Para las descripciones emergentes de cada estación, se han realizado otras dos gráficas independientes. En ellas aparece el valor máximo diario de las magnitudes. Se ha optado por crear dos visualizaciones en vez de una ya que el CO se mide en unidades diferentes al resto de magnitudes. Al incluir estas gráficas en la descripción de una estación, automáticamente se filtra por esa estación y sólo muestra los valores de las magnitudes de ésta. En la figura 33 podemos ver la descripción emergente para la estación 28079024 desde la vista del *dashboard*. Añadir que la descripción emergente aparece cuando se pasa el ratón por encima del punto que simula a la estación.

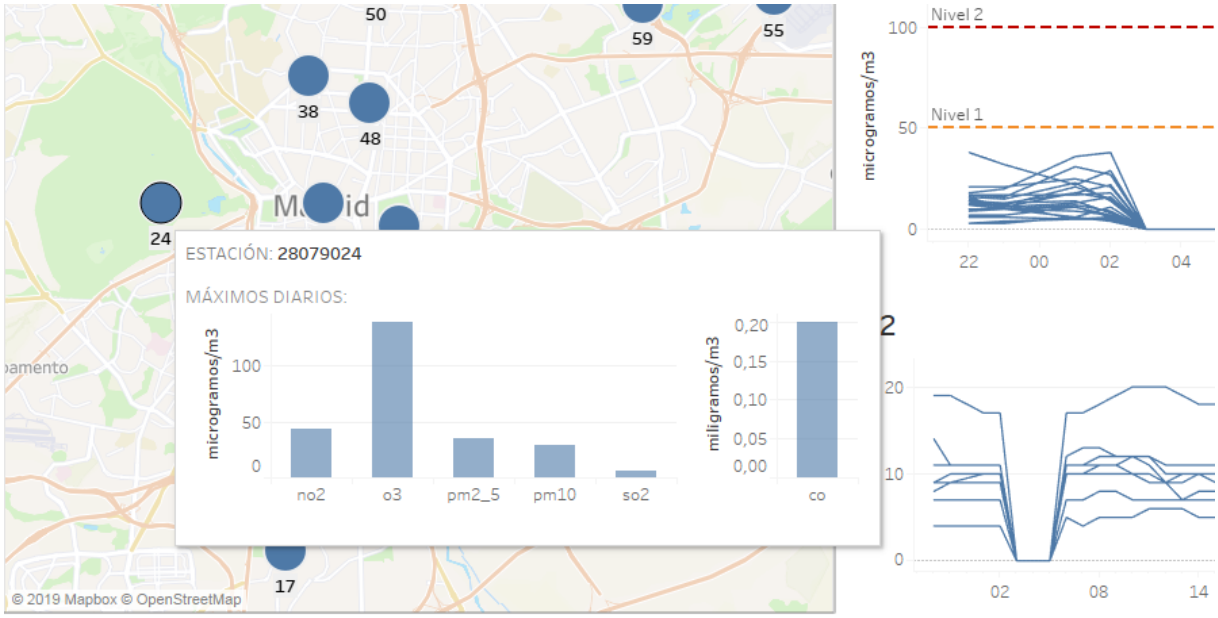


Figura 33: Descripción emergente de una estación en el mapa del *dashboard*

Predicción: indica de los 4 niveles diseñados incialmente en el proyecto la predicción para un determinado día. En el caso del ejemplo las medidas recogidas van desde el día 01/07 al 03/07. Por lo que en la predicción podemos ver el valor para los días 4, 5 y 6 de julio. En el caso de los tres se predice que en todos ellos se alcanzará el nivel 1 de contaminación. El campo calculado que determina si se pinta o no un registro evalúa que la fecha del registro sea posterior al día en el que nos encontramos (*IF (TODAY()<[Fecha]) THEN 'SI' ELSE 'NO' END*), si es posterior el campo tomará el valor de 'SI' y si no lo es, el de 'NO'. Posteriormente en la configuración de la gráfica se indica que para cada día escoja el valor máximo de Predicción. Ya que como comentamos anteriormente, cada vez que se actualizan los datos en tiempo real, se escribe en el fichero una línea con la predicción máxima y la fecha, por lo que a lo largo de un día contaremos con varios registros de predicciones. Finalmente, se añade un filtro en el que sólo se pinten los registros donde el valor Pintar_predicción valga 'SI'. En la figura 34 podemos ver la configuración de la gráfica en Tableau. Utiliza únicamente la fuente de datos prediccion_total.

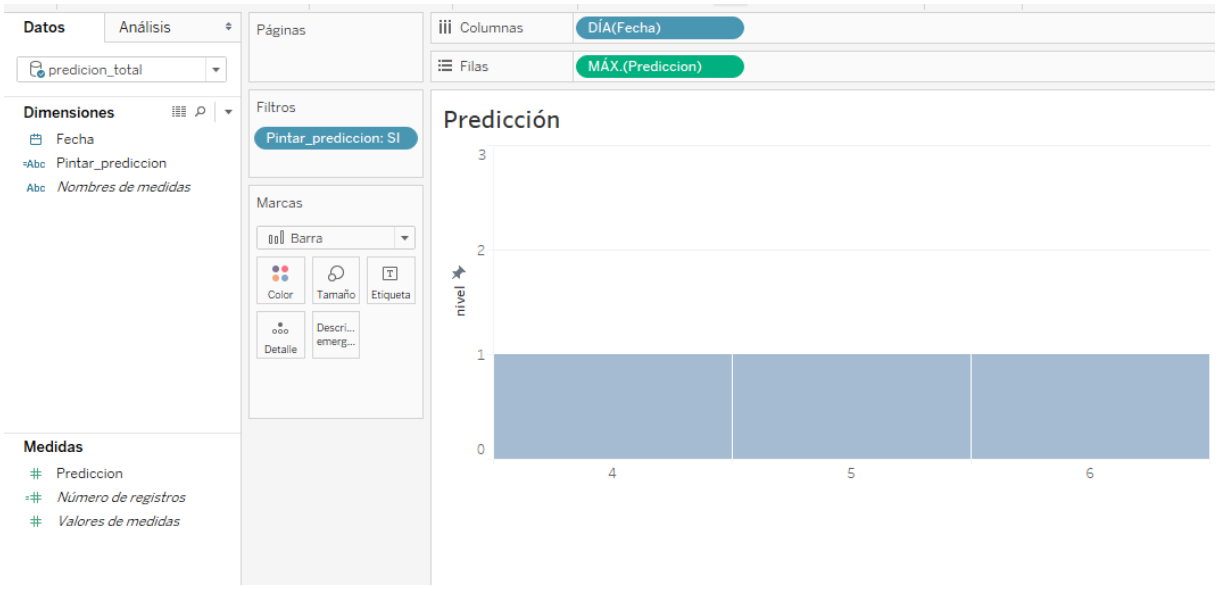


Figura 34: Configuración en Tableau gráfica Predicción del *dashboard*

no_2, so2 y o3: se sirven de los datos de la fuente datos_magnitudes+. Las tres gráficas están realizadas del mismo modo con algunas salvedades. Por ejemplo, en el caso de la gráfica de no_2 se han incluido líneas de referencia que marcan el umbral de los niveles de contaminación. Para la muestra del prototipo del *dasboard* únicamente aparece hasta el nivel 2, simplemente se ha realizado de esta manera para que se pueda visualizar con mayor calidad en las figuras. Una vez pasado a producción aparecería la tercera línea de referencia indicando el umbral del nivel 3. Las tres pintan desde las 22h del día anterior, pero la graduación del eje de las gráficas so2 y o3 es diferente (menor cantidad de números) debido a que en el *dashboard* tienen un tamaño menor.

Para graficar los valores, se han usado los datos de las magnitudes y campos calculados que limitan los datos a pintar (acotan el rango desde las 22h del día anterior hasta el momento actual) además de las estaciones y la hora de cada registro. En la figura 35 podemos ver la configuración de la gráfica. Las visualizaciones so2 y o3 se configurarían igual, sin las líneas de referencia y el filtro Magnitud se cambiaría por so2 u o3.

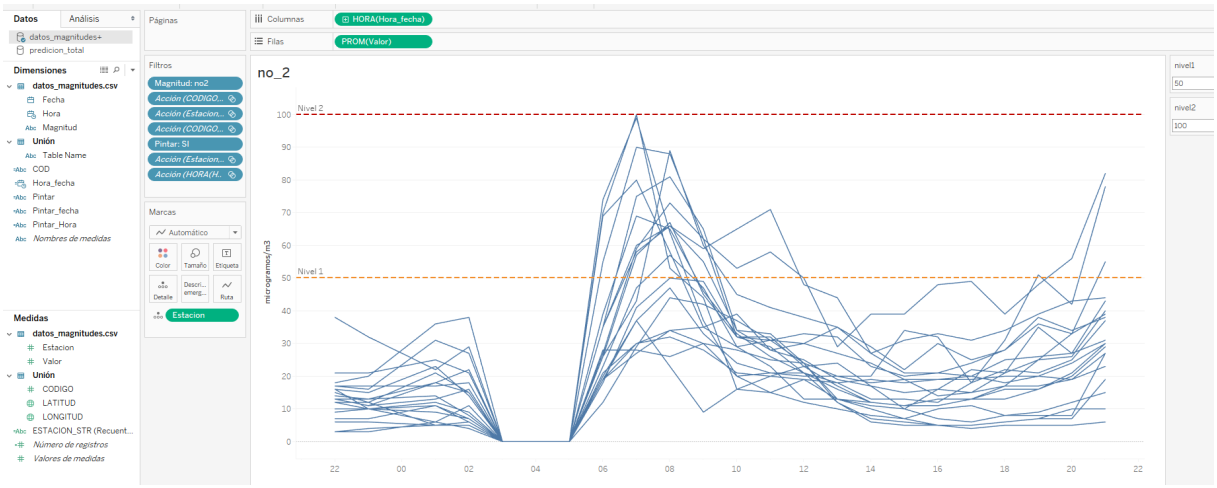


Figura 35: Configuración en Tableau gráfica no_2 del *dashboard*

Al igual que en el mapa, estas tres gráficas cuentan con descripción emergente en las que aparece la estación, la hora y el valor que toma la medida. En la figura 36, desde la vista del *dashboard* podemos ver un ejemplo de como aparecería al pasar el ratón por encima.

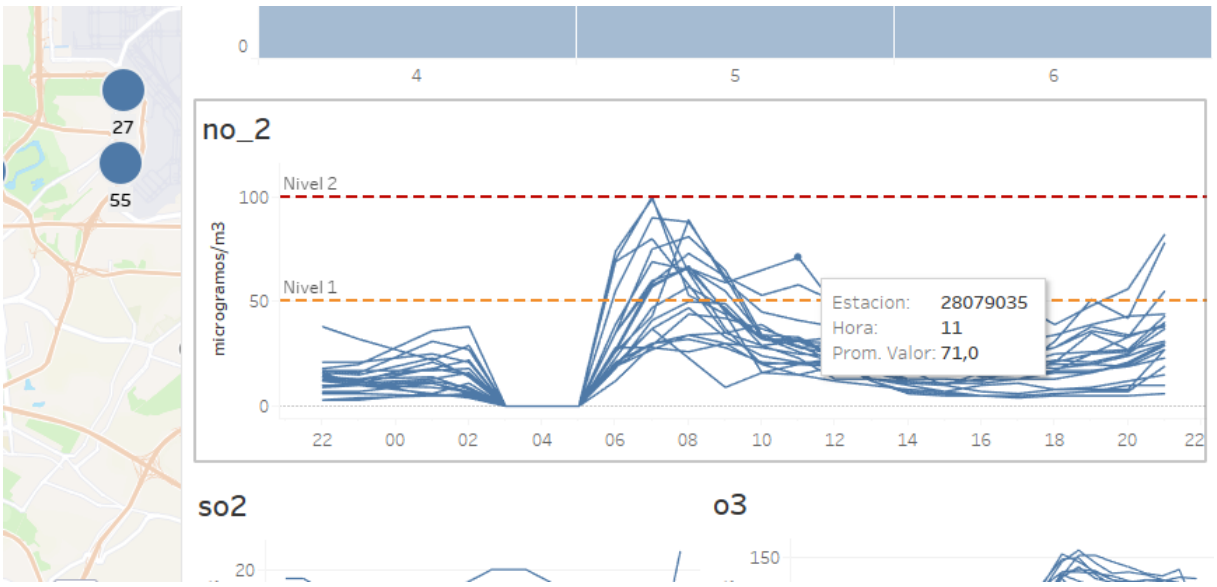


Figura 36: Descripción emergente de una medida en el *dashboard*

***Dashboard*:** cuenta con una interacción de filtrado que conecta todas las visualizaciones salvo la de Predicción. Desde el mapa podríamos activarlo pinchando en una estación en concreto, el resto de gráficas únicamente nos mostrarían la línea que corresponde a dicha estación en caso de contar con datos. Como podemos observar a modo ejemplo con la estación 28079035 en la figura 37

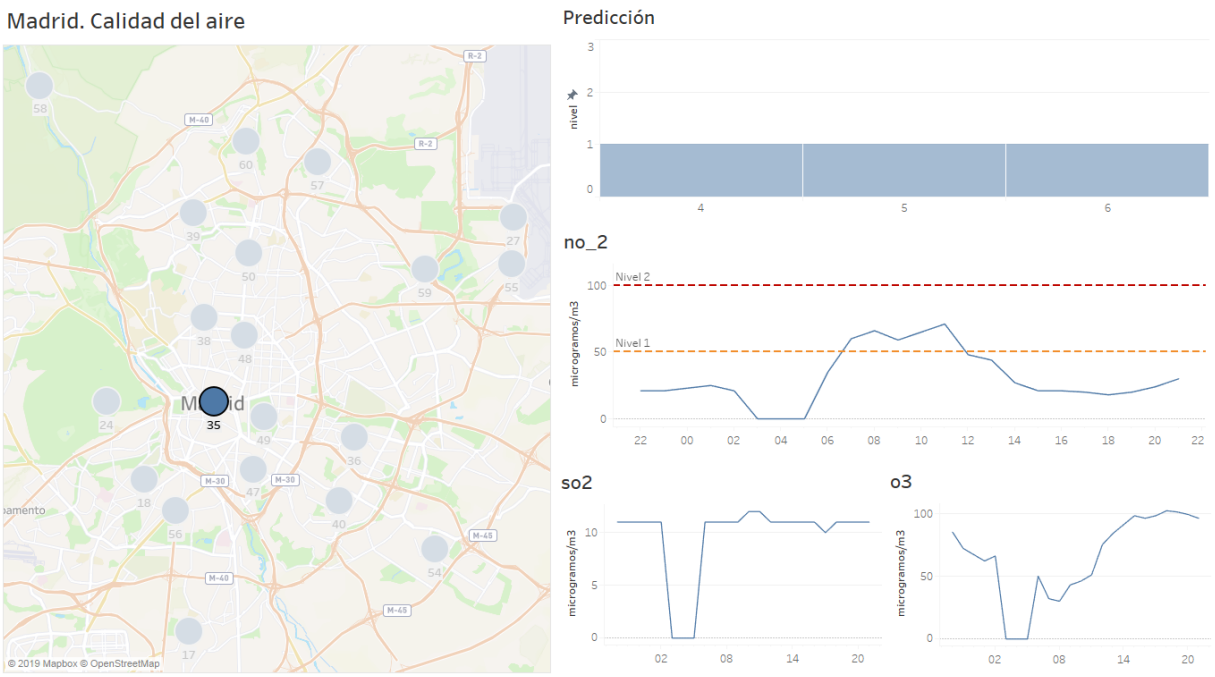


Figura 37: Filtrado desde el mapa en el *dashboard*

Funciona de forma similar desde cualquiera de las otras gráficas. Por ejemplo, en la figura 38 si filtramos por un valor concreto de la gráfica o3 se filtra la estación a la que corresponde en el mapa, en esa gráfica se quedaría únicamente resaltada la línea a la que pertence el punto, pero en las otras gráficas sólo permanecería el valor que toman las otras medidas para esa hora en concreto (se ha señalado con una flecha el punto en cada gráfica).

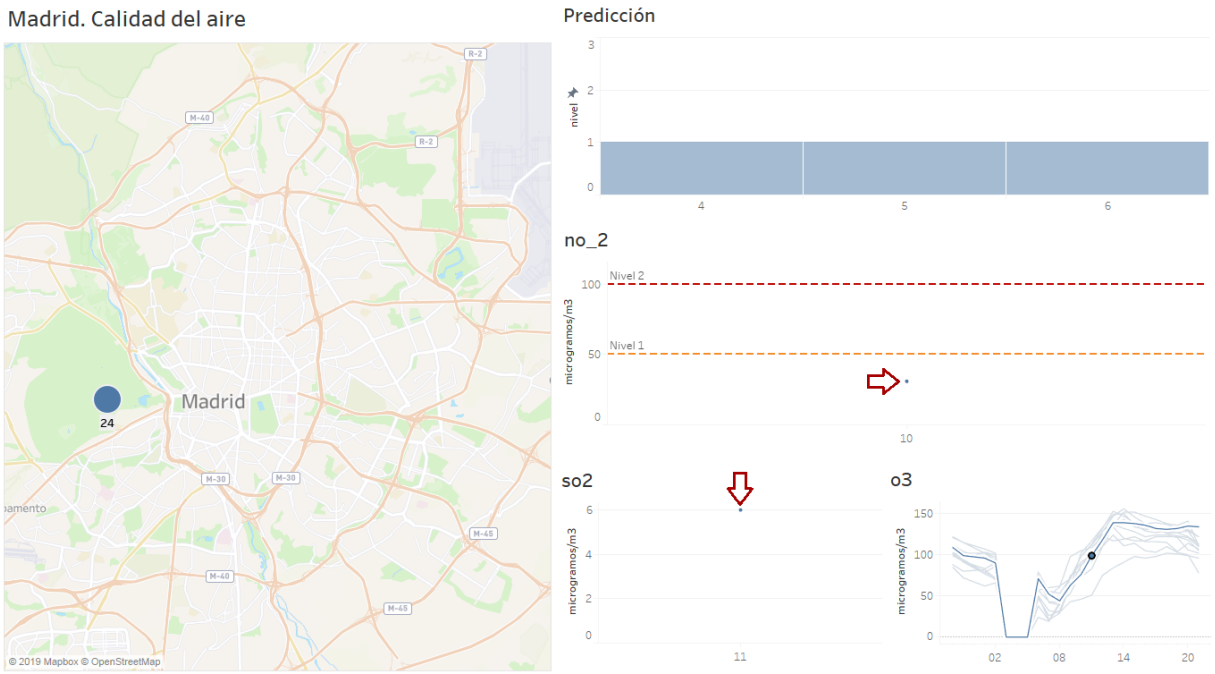


Figura 38: Filtrado desde la gráfica o3 del *dashboard*

Añadir que durante el espacio de tiempo entre las 3 y las 5 se llevan a cabo los trabajos de mantenimiento sobre las estaciones. Durante ese intervalo, los valores de las magnitudes valen 0 en todos los casos. Como se comenta a posteriori en el capítulo 7, en las líneas de futuro, sería recomendable recoger esos datos posteriormente para poder completar las visualizaciones y tener visión del cómputo global de las medidas. Ya que aunque durante ese tiempo las medidas valgan 0, en los *datasets* posteriores proporcionados por el portal esas horas aparecen rellenas con los valores correctos de sus medidas, porque a pesar del mantenimiento las estaciones continúan midiendo.

6. Evaluación

Tras realizar los dos modelos de predicción para cada una de las estaciones de las que contábamos con datos suficientes, 20 en total, se ha procedido a evaluar y compararlos.

Como se ha comentado previamente la parametrización escogida para la creación de los modelos ha sido:

- Árboles: max_depth=profundidad (profundidad máxima que queremos que tenga el árbol, en nuestro caso 6) y criterion = 'entropy' (más enfocado para los valores discretos). Entrenamiento con el 75 % de los datos del conjunto y validación con el 25 %. Podemos ver los parámetros en detalle en la sección 5.4.1, árboles de decisión.
- Redes neuronales: max_iter = 100000 (número máximo de iteraciones), hidden_layer_sizes = (50,25) (número de neuronas en las capas ocultas, en este caso una primera con 50 y una segunda con 25 neuronas). Entrenamiento con el 75 % de los datos del conjunto y validación con el 25 %. Para más información sobre los parámetros se puede consultar la sección 5.4.2, redes neuronales.

Comparamos los resultados (score) obtenidos para cada modelo. En el cuadro 3

ESTACIÓN	MODELO	RESULTADO VAL.	RESULTADO ENTR.
28079017	árbol	0,72254819782062	0,74070969544566
	red	0,73931265716681	0,73232746577256
28079018	árbol	0,74526453298498	0,76964946657958
	red	0,75310254735467	0,75941650337470
28079024	árbol	0,86049543676662	0,87758208306154
	red	0,86766623207301	0,86888454011742
28079027	árbol	0,74052894924946	0,75929456625357
	red	0,75911365260901	0,74189704480458
28079035	árbol	0,66427640156454	0,70636818083025
	red	0,65710560625815	0,67876548576396
28079036	árbol	0,65697674418605	0,69401378122308
	red	0,68087855297158	0,70047372954350
28079038	árbol	0,67633784655061	0,72387096774194
	red	0,68729851708575	0,71483870967742

28079039	árbol	0,66209150326797	0,70945945945946
	red	0,68562091503268	0,69354838709677
28079040	árbol	0,73221216041397	0,75026957084322
	red	0,71733505821475	0,73021350010783
28079047	árbol	0,74289099526066	0,78054567022539
	red	0,76658767772512	0,76433372874654
28079048	árbol	0,72942643391521	0,76830282861897
	red	0,73192019950125	0,71921797004992
28079049	árbol	0,84642857142857	0,86706349206349
	red	0,84047619047619	0,85039682539683
28079050	árbol	0,67560975609756	0,71509971509972
	red	0,67560975609756	0,69474969474969
28079054	árbol	0,79905437352246	0,80710059171598
	red	0,79078014184397	0,78698224852071
28079055	árbol	0,69019138755981	0,74391703230953
	red	0,71411483253589	0,70961308336657
28079056	árbol	0,61278648974668	0,65460394049055
	red	0,65862484921592	0,65299557700040
28079057	árbol	0,78630460448642	0,81739472648564
	red	0,79929161747344	0,80716253443526
28079058	árbol	0,97759433962264	0,99016522423289
	red	0,97995283018868	0,98898505114083
28079059	árbol	0,95011876484561	0,95091053048298
	red	0,95011876484561	0,94893111638955
28079060	árbol	0,80000000000000	0,81785283474065
	red	0,81325301204819	0,82026537997587

Cuadro 3: Para cada estación, resultados por modelo con los datos de validación y de entrenamiento.

A continuación, de forma gráfica, podemos observar en la parte izquierda de la figura 39, el resultado con el conjunto de datos de test, mientras que en la derecha aparecen los obtenidos con los datos de entrenamiento.

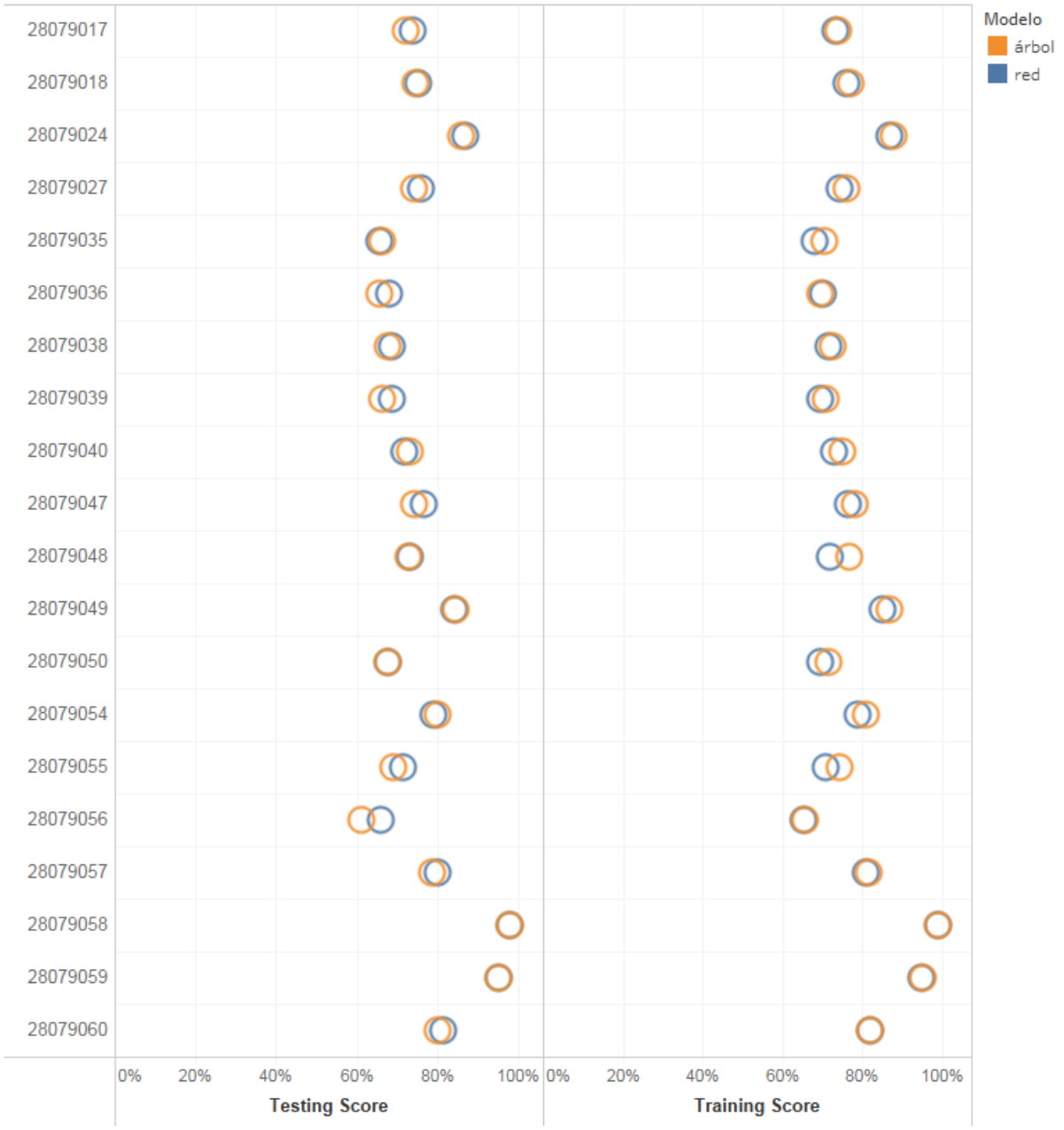


Figura 39: Resultado obtenido para cada modelo con los datos de entrenamiento y de validación

En la figura 39 vemos que en el conjunto de datos de validación están bastante igualados los dos modelos pero exceptuando las estaciones 28079035, 28079040, 28079049 y 28079054 las redes neuronales obtienen mejores resultados. En cambio, en los resultados conseguidos con el conjunto de datos de entrenamiento, para prácticamente todos los casos salvo dos de ellos, el modelo del árbol de descisión ha obtenido mejor resultado.

La matriz de confusión obtenida en las estadíaticas, como hemos visto en la figura 23 o en la figura 26, nos permite visualizar el rendimiento del modelo predictivo, donde se desconocen

los verdaderos valores pudiendo detectar de forma sencilla dónde se está confundiendo el sistema. Está formada por 4 valores: True Positives (TP, clase real y predicha verdadero), True Negatives (TN, clase real y predicha falso), False Positives (FP, clase real falso y predicha verdadero) y False Negatives (FN, clase real verdadero y predicha falso).

Partiendo de la matriz de confusión obtenemos el resto de métricas:

- Precisión: es el porcentaje total de elementos clasificados correctamente. Tiene una gran desventaja y es que si los datos no están balanceados como ocurre en nuestro caso no es la medida más adecuada. En el cuadro 4 donde tenemos la posibilidad de ver el soporte (cantidad de instancias que hay para cada una de las clases) podemos observar como se manifiesta este hecho. Los datos no están equilibrados en la mayoría de los casos y aun así a veces se consiguen valores de precisión bastante altos. Como por ejemplo el caso de la estación 28079058.
- Recall: es la sensibilidad o tasa de True Positive. Nos da la información respecto a los falsos negativos, es decir, cuantos fallaron.
- F1-score: es la medida de precisión empleada en la determinación de un valor único ponderado de la precisión y la exhaustividad (media armónica). Al ser un valor único nos permite comparar dos modelos de predicción entre sí. Por lo que será el valor que deberíamos contemplar por clase, al comparar en el cuadro 4 que modelo de predicción es mejor.

Se recuerda que en la sección 5.4.1, dentro de los árboles de decisión, se explican las métricas con mayor detalle.

ESTACIÓN	MODELO	CLASE	PRECISION	RECALL	F1-SCORE	SUPPORT
28079017	árbol	0	0,77	0,82	0,79	779
		1	0,62	0,53	0,57	414
	red	0	0,77	0,86	0,81	779
		1	0,66	0,52	0,58	414
28079018	árbol	0	0,78	0,87	0,82	1036
		1	0,64	0,48	0,55	495
	red	0	0,78	0,89	0,83	1036
		1	0,66	0,48	0,56	495

28079024	árbol	0	0,89	0,96	0,92	1321
		1	0,5	0,24	0,32	213
	red	0	0,87	1	0,93	1321
		1	0,78	0,07	0,12	213
28079027	árbol	0	0,77	0,89	0,83	983
		1	0,6	0,39	0,47	416
	red	0	0,78	0,91	0,84	983
		1	0,65	0,41	0,51	416
28079035	árbol	0	0,72	0,52	0,6	757
		1	0,63	0,81	0,71	777
	red	0	0,64	0,71	0,67	757
		1	0,68	0,6	0,64	777
28079036	árbol	0	0,68	0,79	0,73	919
		1	0,6	0,46	0,52	629
	red	0	0,69	0,84	0,76	919
		1	0,66	0,45	0,53	629
28079038	árbol	0	0,71	0,65	0,68	808
		1	0,65	0,7	0,68	743
	red	0	0,73	0,64	0,68	808
		1	0,65	0,74	0,69	743
28079039	árbol	0	0,7	0,78	0,74	941
		1	0,57	0,48	0,52	589
	red	0	0,7	0,86	0,77	941
		1	0,65	0,41	0,5	589
28079040	árbol	0	0,76	0,9	0,83	1091
		1	0,58	0,33	0,42	455
	red	0	0,73	0,96	0,83	1091
		1	0,58	0,15	0,23	455
28079047	árbol	0	0,85	0,8	0,82	626
		1	0,5	0,59	0,54	218
	red	0	0,85	0,83	0,84	626
		1	0,54	0,59	0,57	218
28079048	árbol	0	0,77	0,89	0,83	581
		1	0,52	0,3	0,38	221

	red	0	0,83	0,79	0,81	581
		1	0,51	0,57	0,54	221
28079049	árbol	0	0,86	0,97	0,91	708
		1	0,54	0,16	0,25	132
	red	0	0,86	0,97	0,91	708
		1	0,48	0,16	0,24	132
28079050	árbol	0	0,75	0,75	0,75	527
		1	0,55	0,54	0,54	293
	red	0	0,74	0,76	0,75	527
		1	0,55	0,53	0,54	293
28079054	árbol	0	0,82	0,95	0,88	671
		1	0,54	0,22	0,31	175
	red	0	0,83	0,93	0,88	671
		1	0,49	0,25	0,33	175
28079055	árbol	0	0,71	0,89	0,79	544
		1	0,61	0,31	0,41	292
	red	0	0,77	0,8	0,79	544
		1	0,6	0,55	0,57	292
28079056	árbol	0	0,53	0,12	0,19	325
		1	0,62	0,93	0,75	504
	red	0	0,57	0,5	0,54	325
		1	0,7	0,76	0,73	504
28079057	árbol	0	0,85	0,88	0,87	673
		1	0,48	0,41	0,44	174
	red	0	0,81	0,97	0,89	673
		1	0,55	0,13	0,21	174
28079058	árbol	0	0,98	1	0,99	831
		1	0	0	0	17
	red	0	0,98	1	0,99	831
		1	0	0	0	17
28079059	árbol	0	0,95	1	0,97	800
		1	0	0	0	42
	red	0	0,95	1	0,97	800

		1	0	0	0	42
28079060	árbol	0	0,82	0,97	0,89	675
		1	0,26	0,04	0,07	155
	red	0	0,82	0,98	0,9	675
		1	0,5	0,09	0,15	155

Cuadro 4: Para cada estación, resultados por modelo de las diferentes medidas.

En las figuras 40, 41, 42, 43 y 44 podemos observar de forma gráfica los datos del cuadro 4, observando en la parte inferior de cada gráfico el desequilibrio que existe entre la cantidad de instancias de cada clase.

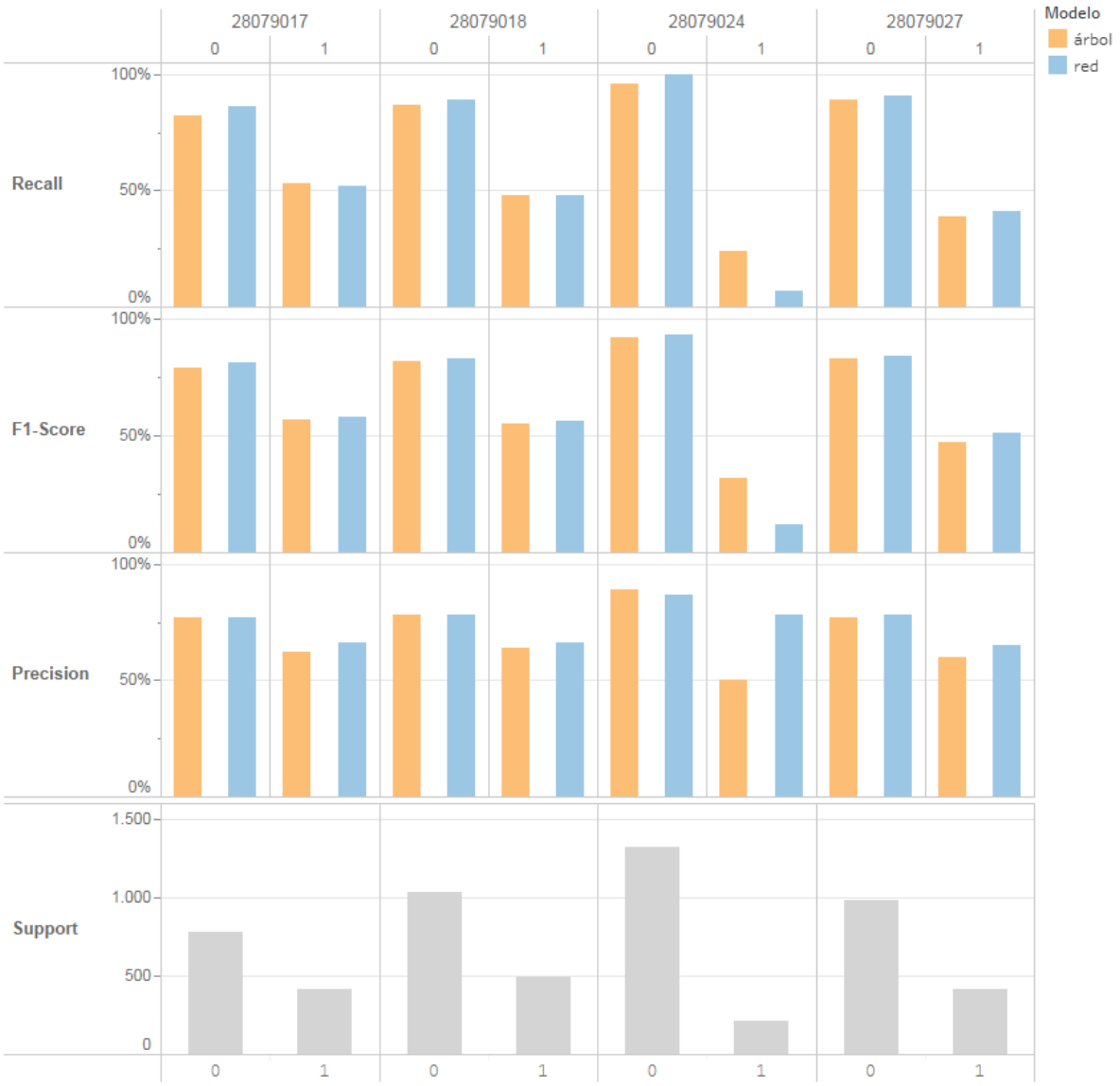


Figura 40: Medidas de las estaciones 28079017, 28079018, 28079024 y 28079023

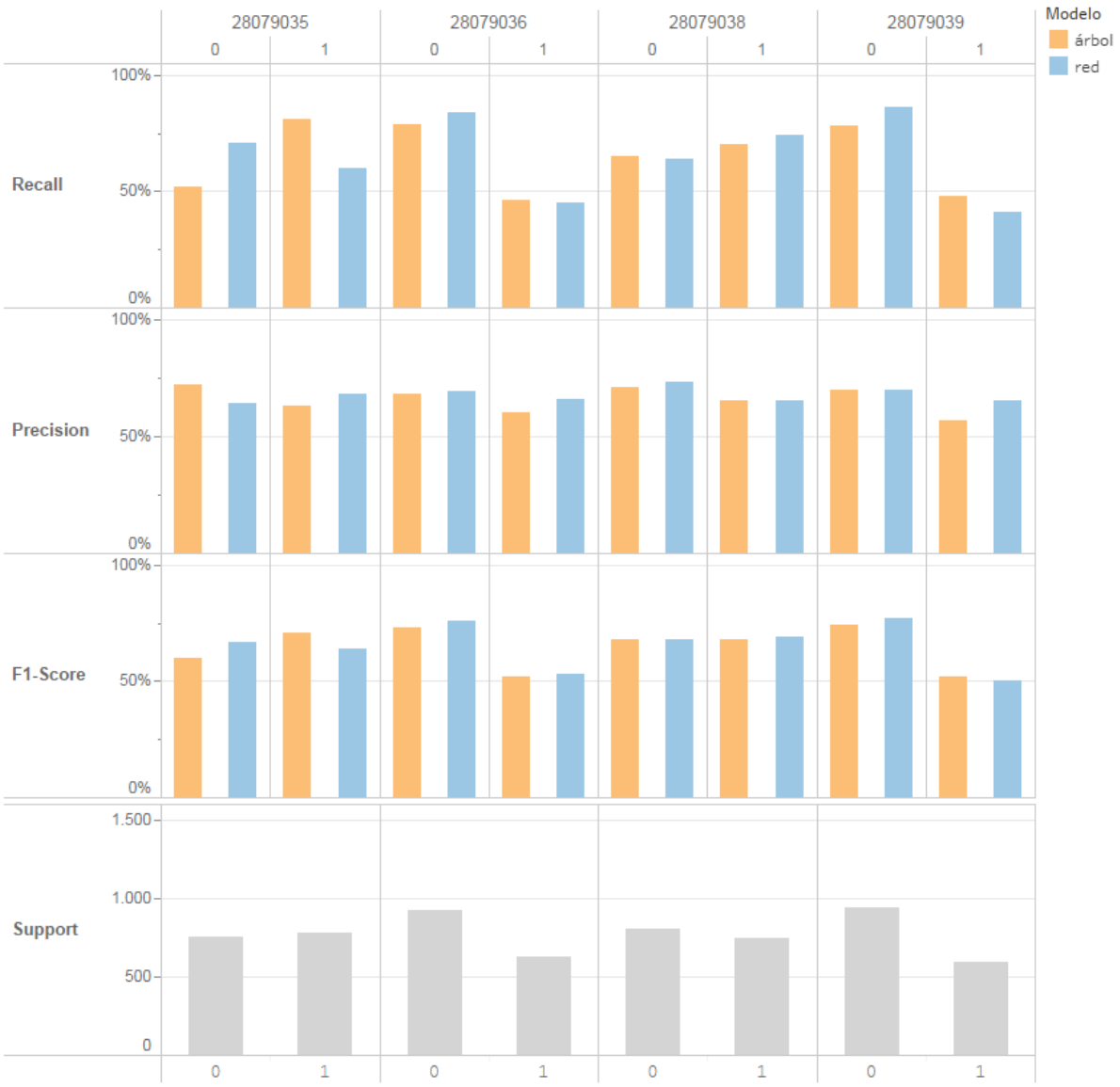


Figura 41: Medidas de las estaciones 28079035, 28079036, 28079038 y 28079039

Figura 42: Medidas de las estaciones 28079040, 28079047, 28079048 y 28079049

Figura 43: Medidas de las estaciones 28079050, 28079054, 28079055 y 28079056

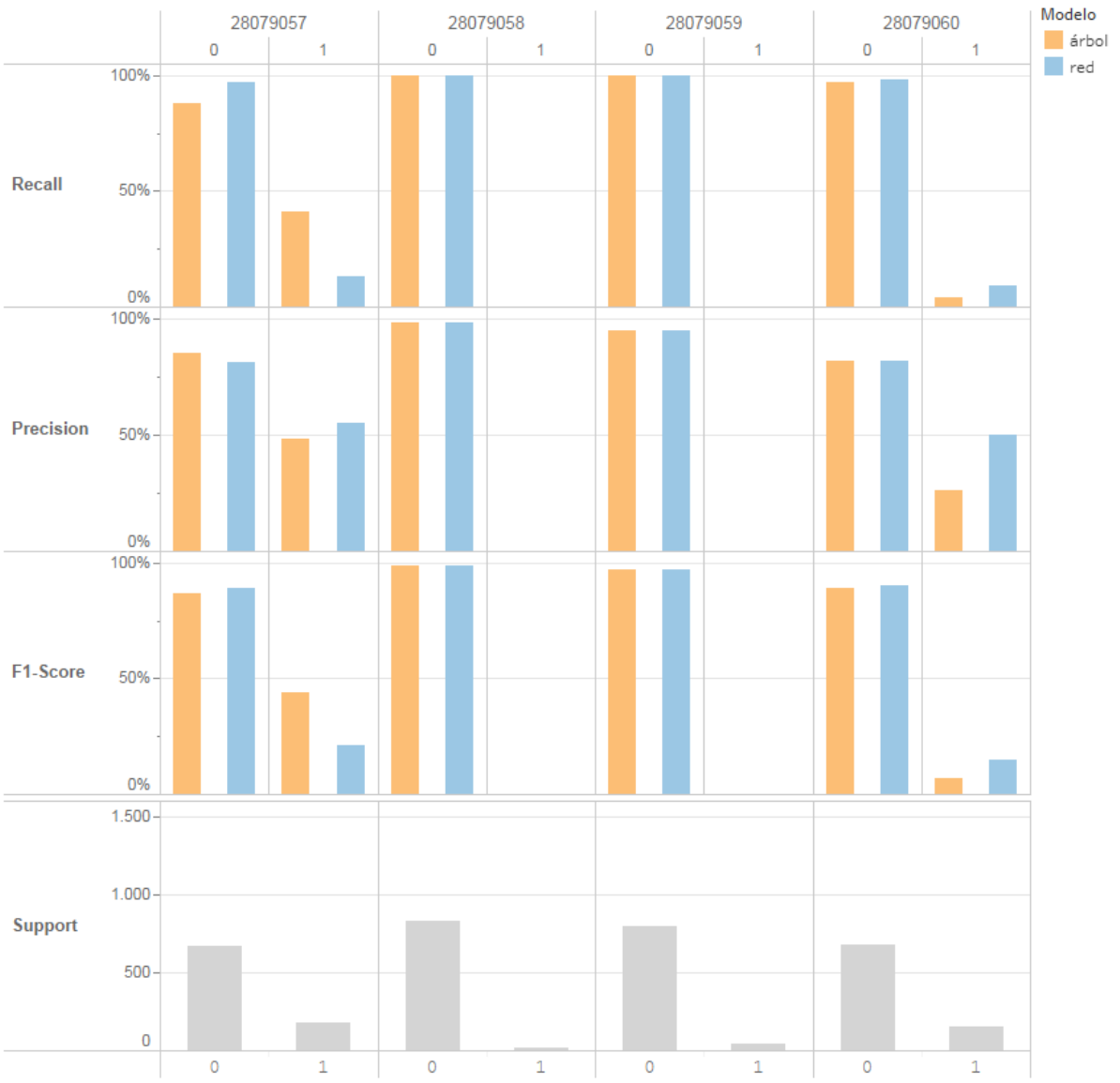


Figura 44: Medidas de las estaciones 28079057, 28079058, 28079059 y 28079060

7. Conclusiones y líneas de futuro

Para cada estación se han creado dos modelos de predicción, un árbol clasificador y una red neuronal. Observando los resultados de las estadísticas, y basándonos en la media armónica (valor f1) que compara los dos modelos para cada una de las clases podemos concluir que el modelo de red neuronal es algo más confiable y detecta mejor las clases. Aunque, en cualquier caso, ambos tienen comportamientos y estadísticas muy similares.

Los resultados no son los esperados ya que si nos marcamos como un umbral satisfactorio el 0,75, podemos observar que el 34 % de los casos tendrían una alta precisión y *recall*, considerando en estos casos que el modelo maneja perfectamente la clase. Sólo el 1 % cuenta con una precisión alta y un *recall* bajo, siendo modelos que no detectan la clase muy bien, pero en el momento que lo hacen son altamente confiables. El 13 % de los casos tienen una precisión baja pero en cambio, el *recall* es alto, quiere decir, que el modelo está detectando bien la clase pero también está incluyendo muestras de la otra clase. Y el 53 % que resta no está llegando al umbral marcado por lo que el modelo en este caso no está logrando clasificar los datos correctamente.

El problema principal de los modelos planteados radica en que el conjunto de datos es muy desequilibrado. De hecho, sólo tenemos muestras para dos de las cuatro clases. Y de las dos clases prácticamente el 70 % de los datos pertenecen a la clase 0 mientras que únicamente el 30 % a la clase 1. Dependiendo de la estación concreta estos valores pueden estar más separados entre sí, como podemos observar en los valores de soporte.

Las mejoras propuestas para llevar a cabo sobre los modelos y conseguir de esta forma mejores resultados serían:

- Ajustar los parámetros del modelo: consistiría en ajustar las métricas y diferentes parámetros del propio algoritmo para intentar equilibrar de esta forma las clases minoritarias penalizando a la clase mayoritaria durante el entrenamiento. Se podría conseguir mediante la librería de Python imbalanced-learn. Para los árboles se podría indicar el parámetro class_weight como *balanced* y en las redes neuronales incluir una métrica de *loss* para penalizar a la clase mayoritaria. Aunque habría que estudiarlo en profundidad ya que no todos los algoritmos cuentan con esta opción.
- Alterar el *dataset*: podríamos eliminar algunas muestras de la clase mayoritaria para reducir las instancias e intentar equilibrar la situación con las clases minoritarias. Aunque

en este caso habría que tener cuidado de no prescindir de muestras importantes que puedan brindar información clave y de esta forma empeorar el modelo. Para no incurrir en este error, habría que seleccionar las muestras a eliminar con algún criterio en concreto. Otra opción sería añadir nuevas filas con los mismos valores de la clase minoritaria, aunque podríamos llegar a tener problemas de *overfitting*.

- Simular muestras: podríamos intentar crear muestras artificiales, no idénticas. Existen diversos algoritmos que intentan seguir la tendencia del grupo minoritario. Habría que tener precaución porque es posible crear muestras sintéticas que alteren la distribución ”natural” de la clase y crear confusión al modelo.

- *Balanced ensemble methods*: tiene la ventaja de realizar un ensamble de métodos, es decir, entrena diversos modelos y entre todos obtiene el resultado final asegurándose de tomar en todo momento muestras de entrenamiento equilibradas.

Respecto a los datos, aparte de los históricos, sería más que recomendable retroalimentar los modelos, es decir, los datos en tiempo real que se van almacenando, cada determinado tiempo, fusionarlos con los históricos usados al inicio y volver a entrenar los modelos con todo el conjunto completo. De esta forma, habrá más instancias de cada clase y quizás los modelos puedan aprender clases que aún desconocen por no contar con ninguna muestra de ellas.

En referencia a los valores de los datos en particular, se propone respecto a la temperatura obtener el valor de la estación climática más próxima a la estación de calidad del aire, y no únicamente tomar el valor de la de Retiro, ya que en Ciudad Universitaria, donde se sitúa la otra estación de medida del clima proporcionará valores más afinados a las estaciones de calidad del aire colindantes.

Habría que revisar las variables que se están teniendo en cuenta y quizás incluir algunas nuevas, como el nivel de predicción, ya que, por ejemplo, si la contaminación supera un determinado umbral dependiendo de la ciudad y la gobernanza, se pueden aplicar medidas restrictivas de tráfico. En este caso, esa acción que se lleva a cabo, puede ser una variable en sí, ya que también influye en la reducción de la contaminación.

Para mejorar el *dashboard* sería necesario desarrollar e implementar una función que pueda recoger los datos de las horas de mantenimiento (aunque fuera a posteriori, unas horas después). Aunque sea un rango temporal donde la contaminación nunca va a tener sus mayores picos, es recomendable porque de esta forma podemos obtener mejores visualizaciones, partiendo de la estética de la gráfica a poder llegar a hacernos una mejor idea de la progresión

de las magnitudes.

Finalmente, la última propuesta para obtener mejor rendimiento es sustituir la actual forma de guardar los datos en ficheros, por una base de datos.

8. Bibliografía

Adame, J.A., Aranda, A., Díaz de Mera, Y., Muñoz, F. Notario, A., Parra, A. Parra, J. y Romero, E. (2005). Proyecto de innovación educativa sobre contaminación atmosférica para alumnos de secundaria. Universidad de Castilla La Mancha. Recuperado de http://uvtnetwork.uclm.es/files/2013/03/Proyecto-de-innovación-educativa-sobre-contaminación-atmosférica.pdf

Alberdi Odriozola, J.C., Díaz Jiménez, J., Mirón Pérez, I.J. y Montero Rubio, J.C. (1997). Influencia de variables atmosféricas sobre la mortalidad por enfermedades respiratorias y cardiovasculares en los mayores de 65 años de la Comunidad de Madrid. Gaceta Sanitaria, volumen(11), 164-170. Recuperado de https://doi.org/10.1016/S0213-9111(97)71294-9

Alberdi Odriozola, J.C., Díaz Jiménez, J., Mirón Pérez, I.J. y Montero Rubio, J.C. (1998). Asociación entre la contaminación atmosférica por dióxido de azufre y partículas totales en suspensión y la mortalidad diaria en la ciudad de Madrid (1986–1992). Gaceta Sanitaria, volumen(12), 207-215. Recuperado de https://doi.org/10.1016/S0213-9111(98)76474-X

Alonso Fustel, E. , Martínez Rueda, T., e. a. (2005). Evaluación en cinco ciudades españolas del impacto en salud de la contaminación atmosférica por partículas. Proyecto europeo Apheis. Revista Española de Salud Pública, volumen(79). Recuperado de https://www.scielosp.org/scielo.php?pid=S1135-57272005000200015&script=sci_arttext&tlng=es

Baldasano, J.M., Ballester F., Medina, S., Querol X., y Sunyer J. (2007). Situación actual, prioridades de actuación y necesidades de investigación en contaminación atmosférica y salud en España: conclusiones del Taller AIRNET de Barcelona. Gaceta Sanitaria, volumen(21). Recuperado de https://www.scielosp.org/article/gs/2007.v21n1/70-75/es/

Bautista Ruiz, L. (2019). Impacto cardiovascular del $PM_{2,5}$ procedente de las emisiones de las centrales térmicas de carbón en España durante el año 2014. Medicina Clínica. Recuperado de https://doi.org/10.1016/j.medcli.2018.11.017

Fernández García, F. (2001). Clima y calidad ambiental en las ciudades: propuesta metodológica y su ampliación al área de Madrid. Proyectos y métodos actuales en Climatología, 41-66. Recuperado de http://geoclima-uam.es/mediapool/128/1280358/data/calidad_ ambiental.pdf

Laverón, F. y Sáenz de Miera, G. (2017). Cambio climático y contaminación del aire: 5 semejanzas, 5 diferencias y 5 reflexiones. El Confidencial. Recuperado de https://blogs.elconfidencial.com/espana/tribuna/2017-02-18/cambio-climatico-contaminacion-semejanzas-diferencias-reflexiones_1334096/

9. Anexos

main.py (Fragmento de código 26)

```
'''
Created on 12 may. 2019

@author: Luz Maria Martinez
'''
from src.gestion_datos import datasets
from src.ia import trees
from src.ia import neural_networks
import pandas as pd
import time
import numpy as np
import datetime
from datetime import timedelta
import os

if __name__ == '__main__':

    #Descargamos el dataset con la estaciones de control
    datasets.get_estaciones_control()

    #Descargamos los historicos de las estaciones de control
    datasets.get_historicos()

    #Descargamos el historico de la climatologia
    datasets.get_climatologia_historico()

    #Procesamos el fichero de las estaciones de control y obtenemos el listado
        de estaciones
    estaciones = datasets.tratar_dataset_estaciones_control()

    #Procesamos el archivo historico de la climatologia
```

```
31    datasets.tratar_dataset_climatologia_historico()
32
33    #Procesamos los historicos de las estaciones de calidad del aire
34    #e incluimos las variables climatologicas procesadas previamente
35    datasets.tratar_dataset_historicos_training()
36
37    #Creamos los inputs para los clasificadores
38    datasets.crear_input_clasificadores_dataset()
39
40
41    #Generamos los arboles para cada estacion
42    (arbol_28079017, score_arbol_28079017) = trees.generar_arbol (28079017, 6,
          True)
43    (arbol_28079018, score_arbol_28079018) = trees.generar_arbol (28079018, 6,
          True)
44    (arbol_28079024, score_arbol_28079024) = trees.generar_arbol (28079024, 6,
          True)
45    (arbol_28079027, score_arbol_28079027) = trees.generar_arbol (28079027, 6,
          True)
46    (arbol_28079035, score_arbol_28079035) = trees.generar_arbol (28079035, 6,
          True)
47    (arbol_28079036, score_arbol_28079036) = trees.generar_arbol (28079036, 6,
          True)
48    (arbol_28079038, score_arbol_28079038) = trees.generar_arbol (28079038, 6,
          True)
49    (arbol_28079039, score_arbol_28079039) = trees.generar_arbol (28079039, 6,
          True)
50    (arbol_28079040, score_arbol_28079040) = trees.generar_arbol (28079040, 6,
          True)
51    (arbol_28079047, score_arbol_28079047) = trees.generar_arbol (28079047, 6,
          True)
52    (arbol_28079048, score_arbol_28079048) = trees.generar_arbol (28079048, 6,
          True)
53    (arbol_28079049, score_arbol_28079049) = trees.generar_arbol (28079049, 6,
```

```
        True)
54  (arbol_28079050, score_arbol_28079050) = trees.generar_arbol (28079050, 6,
        True)
55  (arbol_28079054, score_arbol_28079054) = trees.generar_arbol (28079054, 6,
        True)
56  (arbol_28079055, score_arbol_28079055) = trees.generar_arbol (28079055, 6,
        True)
57  (arbol_28079056, score_arbol_28079056) = trees.generar_arbol (28079056, 6,
        True)
58  (arbol_28079057, score_arbol_28079057) = trees.generar_arbol (28079057, 6,
        True)
59  (arbol_28079058, score_arbol_28079058) = trees.generar_arbol (28079058, 6,
        True)
60  (arbol_28079059, score_arbol_28079059) = trees.generar_arbol (28079059, 6,
        True)
61  (arbol_28079060, score_arbol_28079060) = trees.generar_arbol (28079060, 6,
        True)
62
63
64  #Generamos las redes para cada estacion
65  (red_28079017, score_red_28079017) = neural_networks.generar_red_neuronal
        (28079017, True)
66  (red_28079018, score_red_28079018) = neural_networks.generar_red_neuronal
        (28079018, True)
67  (red_28079024, score_red_28079024) = neural_networks.generar_red_neuronal
        (28079024, True)
68  (red_28079027, score_red_28079027) = neural_networks.generar_red_neuronal
        (28079027, True)
69  (red_28079035, score_red_28079035) = neural_networks.generar_red_neuronal
        (28079035, True)
70  (red_28079036, score_red_28079036) = neural_networks.generar_red_neuronal
        (28079036, True)
71  (red_28079038, score_red_28079038) = neural_networks.generar_red_neuronal
        (28079038, True)
```

```
72    (red_28079039, score_red_28079039) = neural_networks.generar_red_neuronal
          (28079039, True)
73    (red_28079040, score_red_28079040) = neural_networks.generar_red_neuronal
          (28079040, True)
74    (red_28079047, score_red_28079047) = neural_networks.generar_red_neuronal
          (28079047, True)
75    (red_28079048, score_red_28079048) = neural_networks.generar_red_neuronal
          (28079048, True)
76    (red_28079049, score_red_28079049) = neural_networks.generar_red_neuronal
          (28079049, True)
77    (red_28079050, score_red_28079050) = neural_networks.generar_red_neuronal
          (28079050, True)
78    (red_28079054, score_red_28079054) = neural_networks.generar_red_neuronal
          (28079054, True)
79    (red_28079055, score_red_28079055) = neural_networks.generar_red_neuronal
          (28079055, True)
80    (red_28079056, score_red_28079056) = neural_networks.generar_red_neuronal
          (28079056, True)
81    (red_28079057, score_red_28079057) = neural_networks.generar_red_neuronal
          (28079057, True)
82    (red_28079058, score_red_28079058) = neural_networks.generar_red_neuronal
          (28079058, True)
83    (red_28079059, score_red_28079059) = neural_networks.generar_red_neuronal
          (28079059, True)
84    (red_28079060, score_red_28079060) = neural_networks.generar_red_neuronal
          (28079060, True)
85
86    #Nos quedamos con el clasficador que tenga mejor score
87    if score_arbol_28079017 > score_red_28079017:
88        pred_28079017 = arbol_28079017
89    else:
90        pred_28079017 = red_28079017
91
92    if score_arbol_28079018 > score_red_28079018:
```

```
93          pred_28079018 = arbol_28079018
94      else:
95          pred_28079018 = red_28079018
96
97      if score_arbol_28079024 > score_red_28079024:
98          pred_28079024 = arbol_28079024
99      else:
100         pred_28079024 = red_28079024
101
102     if score_arbol_28079027 > score_red_28079027:
103         pred_28079027 = arbol_28079027
104     else:
105         pred_28079027 = red_28079027
106
107     if score_arbol_28079035 > score_red_28079035:
108         pred_28079035 = arbol_28079035
109     else:
110         pred_28079035 = red_28079035
111
112     if score_arbol_28079036 > score_red_28079036:
113         pred_28079036 = arbol_28079036
114     else:
115         pred_28079036 = red_28079036
116
117     if score_arbol_28079038 > score_red_28079038:
118         pred_28079038 = arbol_28079038
119     else:
120         pred_28079038 = red_28079038
121
122     if score_arbol_28079039 > score_red_28079039:
123         pred_28079039 = arbol_28079039
124     else:
125         pred_28079039 = red_28079039
126
```

```
127     if score_arbol_28079040 > score_red_28079040:
128         pred_28079040 = arbol_28079040
129     else:
130         pred_28079040 = red_28079040
131 
132     if score_arbol_28079047 > score_red_28079047:
133         pred_28079047 = arbol_28079047
134     else:
135         pred_28079047 = red_28079047
136 
137     if score_arbol_28079048 > score_red_28079048:
138         pred_28079048 = arbol_28079048
139     else:
140         pred_28079048 = red_28079048
141 
142     if score_arbol_28079049 > score_red_28079049:
143         pred_28079049 = arbol_28079049
144     else:
145         pred_28079049 = red_28079049
146 
147     if score_arbol_28079050 > score_red_28079050:
148         pred_28079050 = arbol_28079050
149     else:
150         pred_28079050 = red_28079050
151 
152     if score_arbol_28079054 > score_red_28079054:
153         pred_28079054 = arbol_28079054
154     else:
155         pred_28079054 = red_28079054
156 
157     if score_arbol_28079055 > score_red_28079055:
158         pred_28079055 = arbol_28079055
159     else:
160         pred_28079055 = red_28079055
```

```
161
162     if score_arbol_28079056 > score_red_28079056:
163         pred_28079056 = arbol_28079056
164     else:
165         pred_28079056 = red_28079056
166
167     if score_arbol_28079057 > score_red_28079057:
168         pred_28079057 = arbol_28079057
169     else:
170         pred_28079057 = red_28079057
171
172     if score_arbol_28079058 > score_red_28079058:
173         pred_28079058 = arbol_28079058
174     else:
175         pred_28079058 = red_28079058
176
177     if score_arbol_28079059 > score_red_28079059:
178         pred_28079059 = arbol_28079059
179     else:
180         pred_28079059 = red_28079059
181
182     if score_arbol_28079060 > score_red_28079060:
183         pred_28079060 = arbol_28079060
184     else:
185         pred_28079060 = red_28079060
186
187
188     if os.path.exists("../data/dashboard") == False:
189         os.mkdir("../data/dashboard")
190
191     print("\n")
192     while True:
193
194         try:
```

```
195         url = datasets.tratar_dataset_tiempo_real()
196         (temp, viento, precip) =
                datasets.tratar_dataset_climatologia_tiempo_real()
197
198
199         cab = True
200         cab3 = True
201         fw=open("../data/dashboard/datos_magnitudes.csv", "a")
202         fw.close()
203         fw3=open("../data/dashboard/predicion_total.csv", "a")
204         fw3.close()
205
206         fr=open("../data/dashboard/datos_magnitudes.csv")
207         contenido = fr.read()
208         if contenido==’’:
209             cab = False
210         fr.close()
211         fr=open("../data/dashboard/datos_magnitudes.csv")
212         contenido = fr.read()
213         if contenido==’’:
214             cab3 = False
215         fr.close()
216
217         fw=open("../data/dashboard/datos_magnitudes.csv", "a")
218         fw2=open("../data/dashboard/predicciones.csv", "w")
219         fw3=open("../data/dashboard/predicion_total.csv", "a")
220
221
222         if cab == False:
223             fw.write(’ESTACION;MAGNITUD;VALOR;FECHA;HORA\n’)
224             cab = True
225
226
227         datos = pd.read_csv (url, delimiter=";")
```

```
228             df = pd.DataFrame(datos)
229             df = df.sort_values(['ESTACION'], ascending=[True])
230
231             try:
232                 df2 = df[df['ESTACION'] == 28079017]
233                 so2 = str(df2[df2['MAGNITUD'] == 'so2']['VALOR'].tolist()[0])
234                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
235                 o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
236                 fecha = str(df2[df2['MAGNITUD'] == 'so2']['FECHA'].tolist()[0])
237                 hora = str(df2[df2['MAGNITUD'] == 'so2']['HORA'].tolist()[0])
238                 fw.write("28079017;so2;"+so2+";"+fecha+";"+hora+"\n")
239                 fw.write("28079017;no2;"+no2+";"+fecha+";"+hora+"\n")
240                 fw.write("28079017;o3;"+o3+";"+fecha+";"+hora+"\n")
241
242                 #Prediccion
243                 atributos = [float(so2),float(no2),float(o3),
244                 float(temp),float(viento),float(precip)]
245                 l_atributos=[]
246                 l_atributos.append(atributos)
247                 l_atributos = np.array(l_atributos, "float32")
248                 pr = pred_28079017.predict(l_atributos)
249                 prediccion = float(pr[0])
250                 fw2.write("28079017;" + str(float(pr[0])) + '\n')
251                 del l_atributos
252                 del atributos
253
254             except IndexError:
255                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079017))
256
257             try:
258                 df2 = df[df['ESTACION'] == 28079018]
259                 so2 = str(df2[df2['MAGNITUD'] == 'so2']['VALOR'].tolist()[0])
260                 co = str(df2[df2['MAGNITUD'] == 'co']['VALOR'].tolist()[0])
```

```
261             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
262             pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
263             o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
264             fecha = str(df2[df2['MAGNITUD'] == 'so2']['FECHA'].tolist()[0])
265             hora = str(df2[df2['MAGNITUD'] == 'so2']['HORA'].tolist()[0])
266             fw.write("28079018;so2;"+so2+";"+fecha+";"+hora+"\n")
267             fw.write("28079018;co;"+co+";"+fecha+";"+hora+"\n")
268             fw.write("28079018;no2;"+no2+";"+fecha+";"+hora+"\n")
269             fw.write("28079018;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
270             fw.write("28079018;o3;"+o3+";"+fecha+";"+hora+"\n")
271
272             #prediccion
273             atributos = [float(so2),float(co),float(no2),float(pm2_5),
274             float(o3),float(temp),float(viento),float(precip)]
275             l_atributos=[]
276             l_atributos.append(atributos)
277             l_atributos = np.array(l_atributos, "float32")
278             pr = pred_28079018.predict(l_atributos)
279             if float(pr[0]) > float(prediccion):
280                 prediccion = float(pr[0])
281             fw2.write("28079018;" + str(float(pr[0])) + '\n')
282             del l_atributos
283             del atributos
284
285         except IndexError:
286             print("--> WARNING: Sin medidas para la estacion " +
                str(28079018))
287
288
289         try:
290             df2 = df[df['ESTACION'] == 28079024]
291             so2 = str(df2[df2['MAGNITUD'] == 'so2']['VALOR'].tolist()[0])
292             co = str(df2[df2['MAGNITUD'] == 'co']['VALOR'].tolist()[0])
293             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
```

```
294             pm10 = str(df2[df2['MAGNITUD'] == 'pm10']['VALOR'].tolist()[0])
295             pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
296             o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
297             fecha = str(df2[df2['MAGNITUD'] == 'so2']['FECHA'].tolist()[0])
298             hora = str(df2[df2['MAGNITUD'] == 'so2']['HORA'].tolist()[0])
299             fw.write("28079024;so2;"+so2+";"+fecha+";"+hora+"\n")
300             fw.write("28079024;co;"+co+";"+fecha+";"+hora+"\n")
301             fw.write("28079024;no2;"+no2+";"+fecha+";"+hora+"\n")
302             fw.write("28079024;pm10;"+pm10+";"+fecha+";"+hora+"\n")
303             fw.write("28079024;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
304             fw.write("28079024;o3;"+o3+";"+fecha+";"+hora+"\n")
305
306             #prediccion
307             atributos = [float(so2),float(co),float(no2),float(pm10),
308             float(pm2_5),float(o3),float(temp),float(viento),
309             float(precip)]
310             l_atributos=[]
311             l_atributos.append(atributos)
312             l_atributos = np.array(l_atributos, "float32")
313             pr = pred_28079024.predict(l_atributos)
314             if float(pr[0]) > float(prediccion):
315                 prediccion = float(pr[0])
316             fw2.write("28079024;" + str(float(pr[0])) + '\n')
317             del l_atributos
318             del atributos
319
320         except IndexError:
321             print("--> WARNING: Sin medidas para la estacion " +
                    str(28079024))
322
323
324         try:
325             df2 = df[df['ESTACION'] == 28079027]
326             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
```

```
327             o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
328             fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
329             hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
330             fw.write("28079027;no2;"+no2+";"+fecha+";"+hora+"\n")
331             fw.write("28079027;o3;"+o3+";"+fecha+";"+hora+"\n")
332
333             #prediccion
334             atributos = [float(no2),float(o3),float(temp),
335             float(viento),float(precip)]
336             l_atributos=[]
337             l_atributos.append(atributos)
338             l_atributos = np.array(l_atributos, "float32")
339             pr = pred_28079027.predict(l_atributos)
340             if float(pr[0]) > float(prediccion):
341                 prediccion = float(pr[0])
342             fw2.write("28079027;" + str(float(pr[0])) + '\n')
343             del l_atributos
344             del atributos
345
346         except IndexError:
347             print("--> WARNING: Sin medidas para la estacion " +
                str(28079027))
348
349
350         try:
351             df2 = df[df['ESTACION'] == 28079035]
352             so2 = str(df2[df2['MAGNITUD'] == 'so2']['VALOR'].tolist()[0])
353             co = str(df2[df2['MAGNITUD'] == 'co']['VALOR'].tolist()[0])
354             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
355             o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
356             fecha = str(df2[df2['MAGNITUD'] == 'so2']['FECHA'].tolist()[0])
357             hora = str(df2[df2['MAGNITUD'] == 'so2']['HORA'].tolist()[0])
358             fw.write("28079035;so2;"+so2+";"+fecha+";"+hora+"\n")
359             fw.write("28079035;co;"+co+";"+fecha+";"+hora+"\n")
```

```
360                 fw.write("28079035;no2;"+no2+";"+fecha+";"+hora+"\n")
361                 fw.write("28079035;o3;"+o3+";"+fecha+";"+hora+"\n")
362
363                 #prediccion
364                 atributos = [float(so2),float(co),float(no2),float(o3),
365                 float(temp),float(viento),float(precip)]
366                 l_atributos=[]
367                 l_atributos.append(atributos)
368                 l_atributos = np.array(l_atributos, "float32")
369                 pr = pred_28079035.predict(l_atributos)
370                 if float(pr[0]) > float(prediccion):
371                     prediccion = float(pr[0])
372                 fw2.write("28079035;" + str(float(pr[0])) + '\n')
373                 del l_atributos
374                 del atributos
375
376             except IndexError:
377                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079035))
378
379             try:
380                 df2 = df[df['ESTACION'] == 28079036]
381                 so2 = str(df2[df2['MAGNITUD'] == 'so2']['VALOR'].tolist()[0])
382                 co = str(df2[df2['MAGNITUD'] == 'co']['VALOR'].tolist()[0])
383                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
384                 pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
385                 fecha = str(df2[df2['MAGNITUD'] == 'so2']['FECHA'].tolist()[0])
386                 hora = str(df2[df2['MAGNITUD'] == 'so2']['HORA'].tolist()[0])
387                 fw.write("28079036;so2;"+so2+";"+fecha+";"+hora+"\n")
388                 fw.write("28079036;co;"+co+";"+fecha+";"+hora+"\n")
389                 fw.write("28079036;no2;"+no2+";"+fecha+";"+hora+"\n")
390                 fw.write("28079036;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
391
392                 #prediccion
```

```
393                 atributos = [float(so2),float(co),float(no2),float(pm2_5),
394                 float(temp),float(viento),float(precip)]
395                 l_atributos=[]
396                 l_atributos.append(atributos)
397                 l_atributos = np.array(l_atributos, "float32")
398                 pr = pred_28079036.predict(l_atributos)
399                 if float(pr[0]) > float(prediccion):
400                     prediccion = float(pr[0])
401                 fw2.write("28079036;" + str(float(pr[0])) + '\n')
402                 del l_atributos
403                 del atributos
404 
405             except IndexError:
406                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079036))
407 
408             try:
409                 df2 = df[df['ESTACION'] == 28079038]
410                 so2 = str(df2[df2['MAGNITUD'] == 'so2']['VALOR'].tolist()[0])
411                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
412                 pm10 = str(df2[df2['MAGNITUD'] == 'pm10']['VALOR'].tolist()[0])
413                 pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
414                 fecha = str(df2[df2['MAGNITUD'] == 'so2']['FECHA'].tolist()[0])
415                 hora = str(df2[df2['MAGNITUD'] == 'so2']['HORA'].tolist()[0])
416                 fw.write("28079038;so2;"+so2+";"+fecha+";"+hora+"\n")
417                 fw.write("28079038;no2;"+no2+";"+fecha+";"+hora+"\n")
418                 fw.write("28079038;pm10;"+pm10+";"+fecha+";"+hora+"\n")
419                 fw.write("28079038;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
420 
421 
422                 #prediccion
423                 atributos = [float(so2),float(no2),float(pm10),float(pm2_5),
424                 float(temp),float(viento),float(precip)]
425                 l_atributos=[]
```

```
426                 l_atributos.append(atributos)
427                 l_atributos = np.array(l_atributos, "float32")
428                 pr = pred_28079038.predict(l_atributos)
429                 if float(pr[0]) > float(prediccion):
430                     prediccion = float(pr[0])
431                 fw2.write("28079038;" + str(float(pr[0])) + '\n')
432                 del l_atributos
433                 del atributos
434 
435             except IndexError:
436                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079038))
437 
438             try:
439                 df2 = df[df['ESTACION'] == 28079039]
440                 co = str(df2[df2['MAGNITUD'] == 'co']['VALOR'].tolist()[0])
441                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
442                 o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
443                 fecha = str(df2[df2['MAGNITUD'] == 'co']['FECHA'].tolist()[0])
444                 hora = str(df2[df2['MAGNITUD'] == 'co']['HORA'].tolist()[0])
445                 fw.write("28079039;co;"+co+";"+fecha+";"+hora+"\n")
446                 fw.write("28079039;no2;"+no2+";"+fecha+";"+hora+"\n")
447                 fw.write("28079039;o3;"+o3+";"+fecha+";"+hora+"\n")
448 
449                 #prediccion
450                 atributos = [float(co),float(no2),float(o3),
451                 float(temp),float(viento),float(precip)]
452                 l_atributos=[]
453                 l_atributos.append(atributos)
454                 l_atributos = np.array(l_atributos, "float32")
455                 pr = pred_28079039.predict(l_atributos)
456                 if float(pr[0]) > float(prediccion):
457                     prediccion = float(pr[0])
458                 fw2.write("28079039;" + str(float(pr[0])) + '\n')
```

```
459             del l_atributos
460             del atributos
461
462         except IndexError:
463             print("--> WARNING: Sin medidas para la estacion " +
                    str(28079039))
464
465         try:
466             df2 = df[df['ESTACION'] == 28079040]
467             so2 = str(df2[df2['MAGNITUD'] == 'so2']['VALOR'].tolist()[0])
468             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
469             pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
470             fecha = str(df2[df2['MAGNITUD'] == 'so2']['FECHA'].tolist()[0])
471             hora = str(df2[df2['MAGNITUD'] == 'so2']['HORA'].tolist()[0])
472             fw.write("28079040;so2;"+so2+";"+fecha+";"+hora+"\n")
473             fw.write("28079040;no2;"+no2+";"+fecha+";"+hora+"\n")
474             fw.write("28079040;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
475
476             #prediccion
477             atributos = [float(so2),float(no2),float(pm2_5),
478             float(temp),float(viento),float(precip)]
479             l_atributos=[]
480             l_atributos.append(atributos)
481             l_atributos = np.array(l_atributos, "float32")
482             pr = pred_28079040.predict(l_atributos)
483             if float(pr[0]) > float(prediccion):
484                 prediccion = float(pr[0])
485             fw2.write("28079040;" + str(float(pr[0])) + '\n')
486             del l_atributos
487             del atributos
488
489         except IndexError:
490             print("--> WARNING: Sin medidas para la estacion " +
                    str(28079040))
```

```
491
492             try:
493                 df2 = df[df['ESTACION'] == 28079047]
494                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
495                 pm10 = str(df2[df2['MAGNITUD'] == 'pm10']['VALOR'].tolist()[0])
496                 pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
497                 fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
498                 hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
499                 fw.write("28079047;no2;"+no2+";"+fecha+";"+hora+"\n")
500                 fw.write("28079047;pm10;"+pm10+";"+fecha+";"+hora+"\n")
501                 fw.write("28079047;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
502
503                 #prediccion
504                 atributos = [float(no2),float(pm10),float(pm2_5),
505                 float(temp),float(viento),float(precip)]
506                 l_atributos=[]
507                 l_atributos.append(atributos)
508                 l_atributos = np.array(l_atributos, "float32")
509                 pr = pred_28079047.predict(l_atributos)
510                 if float(pr[0]) > float(prediccion):
511                     prediccion = float(pr[0])
512                 fw2.write("28079047;" + str(float(pr[0])) + '\n')
513                 del l_atributos
514                 del atributos
515
516             except IndexError:
517                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079047))
518
519             try:
520                 df2 = df[df['ESTACION'] == 28079048]
521                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
522                 pm10 = str(df2[df2['MAGNITUD'] == 'pm10']['VALOR'].tolist()[0])
523                 pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
```

```
524             fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
525             hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
526             fw.write("28079048;no2;"+no2+";"+fecha+";"+hora+"\n")
527             fw.write("28079048;pm10;"+pm10+";"+fecha+";"+hora+"\n")
528             fw.write("28079048;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
529
530             #prediccion
531             atributos = [float(no2),float(pm10),float(pm2_5),
532             float(temp),float(viento),float(precip)]
533             l_atributos=[]
534             l_atributos.append(atributos)
535             l_atributos = np.array(l_atributos, "float32")
536             pr = pred_28079048.predict(l_atributos)
537             if float(pr[0]) > float(prediccion):
538                 prediccion = float(pr[0])
539             fw2.write("28079048;" + str(float(pr[0])) + '\n')
540             del l_atributos
541             del atributos
542
543         except IndexError:
544             print("--> WARNING: Sin medidas para la estacion " +
                str(28079048))
545
546         try:
547             df2 = df[df['ESTACION'] == 28079049]
548             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
549             o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
550             fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
551             hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
552             fw.write("28079049;no2;"+no2+";"+fecha+";"+hora+"\n")
553             fw.write("28079049;o3;"+o3+";"+fecha+";"+hora+"\n")
554
555             #prediccion
556             atributos = [float(no2),float(o3),float(temp),
```

```
557                 float(viento),float(precip)]
558                 l_atributos=[]
559                 l_atributos.append(atributos)
560                 l_atributos = np.array(l_atributos, "float32")
561                 pr = pred_28079049.predict(l_atributos)
562                 if float(pr[0]) > float(prediccion):
563                     prediccion = float(pr[0])
564                 fw2.write("28079049;" + str(float(pr[0])) + '\n')
565                 del l_atributos
566                 del atributos
567
568             except IndexError:
569                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079049))
570
571
572             try:
573                 df2 = df[df['ESTACION'] == 28079050]
574                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
575                 pm10 = str(df2[df2['MAGNITUD'] == 'pm10']['VALOR'].tolist()[0])
576                 pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
577                 fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
578                 hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
579                 fw.write("28079050;no2;"+no2+";"+fecha+";"+hora+"\n")
580                 fw.write("28079050;pm10;"+pm10+";"+fecha+";"+hora+"\n")
581                 fw.write("28079050;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
582
583
584                 #prediccion
585                 atributos = [float(no2),float(pm10),float(pm2_5),
586                 float(temp),float(viento),float(precip)]
587                 l_atributos=[]
588                 l_atributos.append(atributos)
589                 l_atributos = np.array(l_atributos, "float32")
```

```
590             pr = pred_28079050.predict(l_atributos)
591             if float(pr[0]) > float(prediccion):
592                 prediccion = float(pr[0])
593             fw2.write("28079050;" + str(float(pr[0])) + '\n')
594             del l_atributos
595             del atributos
596 
597         except IndexError:
598             print("--> WARNING: Sin medidas para la estacion " +
                    str(28079050))
599 
600         try:
601             df2 = df[df['ESTACION'] == 28079054]
602             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
603             o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
604             fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
605             hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
606             fw.write("28079054;no2;"+no2+";"+fecha+";"+hora+"\n")
607             fw.write("28079054;o3;"+o3+";"+fecha+";"+hora+"\n")
608 
609             #prediccion
610             atributos = [float(no2),float(o3),float(temp),float(viento),
611             float(precip)]
612             l_atributos=[]
613             l_atributos.append(atributos)
614             l_atributos = np.array(l_atributos, "float32")
615             pr = pred_28079054.predict(l_atributos)
616             if float(pr[0]) > float(prediccion):
617                 prediccion = float(pr[0])
618             fw2.write("28079054;" + str(float(pr[0])) + '\n')
619             del l_atributos
620             del atributos
621 
622         except IndexError:
```

```
623             print("--> WARNING: Sin medidas para la estacion " +
                    str(28079054))
624
625         try:
626             df2 = df[df['ESTACION'] == 28079055]
627             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
628             pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
629             fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
630             hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
631             fw.write("28079055;no2;"+no2+";"+fecha+";"+hora+"\n")
632             fw.write("28079055;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
633
634             #prediccion
635             atributos = [float(no2),float(pm2_5),float(temp),
636             float(viento),float(precip)]
637             l_atributos=[]
638             l_atributos.append(atributos)
639             l_atributos = np.array(l_atributos, "float32")
640             pr = pred_28079055.predict(l_atributos)
641             if float(pr[0]) > float(prediccion):
642                 prediccion = float(pr[0])
643             fw2.write("28079055;" + str(float(pr[0])) + '\n')
644             del l_atributos
645             del atributos
646
647         except IndexError:
648             print("--> WARNING: Sin medidas para la estacion " +
                    str(28079055))
649
650         try:
651             df2 = df[df['ESTACION'] == 28079056]
652             co = str(df2[df2['MAGNITUD'] == 'co']['VALOR'].tolist()[0])
653             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
654             o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
```

```
655             fecha = str(df2[df2['MAGNITUD'] == 'co']['FECHA'].tolist()[0])
656             hora = str(df2[df2['MAGNITUD'] == 'co']['HORA'].tolist()[0])
657             fw.write("28079056;co;"+co+";"+fecha+";"+hora+"\n")
658             fw.write("28079056;no2;"+no2+";"+fecha+";"+hora+"\n")
659             fw.write("28079056;o3;"+o3+";"+fecha+";"+hora+"\n")
660
661             #prediccion
662             atributos = [float(co),float(no2),float(o3),
663             float(temp),float(viento),float(precip)]
664             l_atributos=[]
665             l_atributos.append(atributos)
666             l_atributos = np.array(l_atributos, "float32")
667             pr = pred_28079056.predict(l_atributos)
668             if float(pr[0]) > float(prediccion):
669                 prediccion = float(pr[0])
670             fw2.write("28079056;" + str(float(pr[0])) + '\n')
671             del l_atributos
672             del atributos
673
674         except IndexError:
675             print("--> WARNING: Sin medidas para la estacion " +
                 str(28079056))
676
677         try:
678             df2 = df[df['ESTACION'] == 28079057]
679             so2 = str(df2[df2['MAGNITUD'] == 'so2']['VALOR'].tolist()[0])
680             co = str(df2[df2['MAGNITUD'] == 'co']['VALOR'].tolist()[0])
681             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
682             pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
683             fecha = str(df2[df2['MAGNITUD'] == 'so2']['FECHA'].tolist()[0])
684             hora = str(df2[df2['MAGNITUD'] == 'so2']['HORA'].tolist()[0])
685             fw.write("28079057;so2;"+so2+";"+fecha+";"+hora+"\n")
686             fw.write("28079057;co;"+co+";"+fecha+";"+hora+"\n")
687             fw.write("28079057;no2;"+no2+";"+fecha+";"+hora+"\n")
```

```
688             fw.write("28079057;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
689
690             #prediccion
691             atributos = [float(so2),float(co),float(no2),float(pm2_5),
692             float(temp),float(viento),float(precip)]
693             l_atributos=[]
694             l_atributos.append(atributos)
695             l_atributos = np.array(l_atributos, "float32")
696             pr = pred_28079057.predict(l_atributos)
697             if float(pr[0]) > float(prediccion):
698                 prediccion = float(pr[0])
699             fw2.write("28079057;" + str(float(pr[0])) + '\n')
700             del l_atributos
701             del atributos
702
703         except IndexError:
704             print("--> WARNING: Sin medidas para la estacion " +
                    str(28079057))
705
706         try:
707             df2 = df[df['ESTACION'] == 28079058]
708             no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
709             o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
710             fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
711             hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
712             fw.write("28079058;no2;"+no2+";"+fecha+";"+hora+"\n")
713             fw.write("28079058;o3;"+o3+";"+fecha+";"+hora+"\n")
714
715             #prediccion
716             atributos = [float(no2),float(o3),float(temp),
717             float(viento),float(precip)]
718             l_atributos=[]
719             l_atributos.append(atributos)
720             l_atributos = np.array(l_atributos, "float32")
```

```
721                 pr = pred_28079058.predict(l_atributos)
722                 if float(pr[0]) > float(prediccion):
723                     prediccion = float(pr[0])
724                 fw2.write("28079058;" + str(float(pr[0])) + '\n')
725                 del l_atributos
726                 del atributos
727 
728             except IndexError:
729                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079058))
730 
731             try:
732                 df2 = df[df['ESTACION'] == 28079059]
733                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
734                 o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
735                 fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
736                 hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
737                 fw.write("28079059;no2;"+no2+";"+fecha+";"+hora+"\n")
738                 fw.write("28079059;o3;"+o3+";"+fecha+";"+hora+"\n")
739 
740                 #prediccion
741                 atributos = [float(no2),float(o3),float(temp),
742                 float(viento),float(precip)]
743                 l_atributos=[]
744                 l_atributos.append(atributos)
745                 l_atributos = np.array(l_atributos, "float32")
746                 pr = pred_28079059.predict(l_atributos)
747                 if float(pr[0]) > float(prediccion):
748                     prediccion = float(pr[0])
749                 fw2.write("28079059;" + str(float(pr[0])) + '\n')
750                 del l_atributos
751                 del atributos
752 
753             except IndexError:
```

```
754                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079059))
755
756             try:
757                 df2 = df[df['ESTACION'] == 28079060]
758                 no2 = str(df2[df2['MAGNITUD'] == 'no2']['VALOR'].tolist()[0])
759                 pm2_5 = str(df2[df2['MAGNITUD'] == 'pm2_5']['VALOR'].tolist()[0])
760                 o3 = str(df2[df2['MAGNITUD'] == 'o3']['VALOR'].tolist()[0])
761                 fecha = str(df2[df2['MAGNITUD'] == 'no2']['FECHA'].tolist()[0])
762                 hora = str(df2[df2['MAGNITUD'] == 'no2']['HORA'].tolist()[0])
763                 fw.write("28079060;no2;"+no2+";"+fecha+";"+hora+"\n")
764                 fw.write("28079060;pm2_5;"+pm2_5+";"+fecha+";"+hora+"\n")
765                 fw.write("28079060;o3;"+o3+";"+fecha+";"+hora+"\n")
766
767                 #prediccion
768                 atributos = [float(no2),float(pm2_5),float(o3),
769                 float(temp),float(viento),float(precip)]
770                 l_atributos=[]
771                 l_atributos.append(atributos)
772                 l_atributos = np.array(l_atributos, "float32")
773                 pr = pred_28079060.predict(l_atributos)
774                 if float(pr[0]) > float(prediccion):
775                     prediccion = float(pr[0])
776                 fw2.write("28079060;" + str(float(pr[0])) + '\n')
777                 del l_atributos
778                 del atributos
779
780             except IndexError:
781                 print("--> WARNING: Sin medidas para la estacion " +
                        str(28079060))
782
783             if cab3 == False:
784                 fw3.write("Prediccion;Fecha\n")
785                 cab3 = True
```

```
786
787             date = datetime.datetime.strptime(fecha, '%d/%m/%Y').date()
788             date = date + timedelta(days=3)
789             if date.day < 10:
790                 dia = "0" + str(date.day)
791             else:
792                 dia = str(date.day)
793             if date.month < 10:
794                 mes = "0" + str(date.month)
795             else:
796                 mes = str(date.month)
797             anio = str(date.year)
798             fw3.write(str(prediccion)+";"+ dia + "/" + mes + "/" + anio + "\n")
799
800             fw.close()
801             fw2.close()
802             fw3.close()
803
804             localtime = time.asctime( time.localtime(time.time()) )
805             print("--> UPDATE: Datos actualizados " + str(localtime))
806
807             time.sleep(3598)
808
809         except:
810             localtime = time.asctime( time.localtime(time.time()) )
811             print("--> ERROR: No se han podido obtener medidas para actualizar "
                    + str(localtime))
812             time.sleep(300)
```

Fragmento de código 26: main.py

Paquete gestion_datos

datasets.py (Fragmento de código 27)

```
1
2  '''
3  Created on 12 may. 2019
4
5  @author: Luz Maria Martinez
6  '''
7
8
9  from xml.dom import minidom
10 from src.gestion_datos import datos_web
11 import os
12 import requests
13 import json
14 import xlrd
15 import shutil
16 import pandas as pd
17 from src.gestion_datos import datasets
18 from datetime import datetime
19 from lxml import html
20
21
22
23 def get_estaciones_control ():
24     """Descarga el dataset con la estaciones de control de aire, su ubicacion y
           las magnitudes que mide cada una. Lo guarda en disco.
25
26     Excepciones:
27     ValueError -- Si la funcion descargarArchivosUrl propaga algun error.
28
29     """
30
```

```
31      urlDescargar =
            "https://datos.madrid.es/egob/catalogo/212629-0-estaciones-control
32      -aire.xls"
33      urlGuardar =
            "../data/estaciones_control/212629-0-estaciones-control-aire.xls"
34
35      if os.path.exists("../data/estaciones_control") == False:
36          os.mkdir("../data/estaciones_control")
37
38      try:
39          datos_web.descargarArchivosUrl (urlDescargar, urlGuardar, 0)
40      except ValueError as descripcion:
41          print("Ocurrio un error previsto:", descripcion)
42
43
44  def get_historicos ():
45      """Descarga los ficheros con los datos historicos de calidad del aire desde
            el anio 2001 hasta la actualidad
46      Los guarda en disco.
47
48      Devuelve una lista con las ubicaciones de todos los datasets guardados en
            local
49
50      Excepciones:
51      ValueError -- Si la funcion descargarArchivosUrl propaga algun error.
52
53      """
54
55      urlDescargar = 'https://datos.madrid.es/egob/catalogo/201410-0-calidad-aire-
56      diario.dcat'
57      urlGuardar = "../data/historico/201410-0-calidad-aire-diario.dcat"
58
59      if os.path.exists("../data/historico") == False:
60          os.mkdir("../data/historico")
```

```
61
62      lista = []
63
64      try:
65          datos_web.descargarArchivosUrl (urlDescargar, urlGuardar, 0)
66          lurls = []
67          doc = minidom.parse(urlGuardar)
68          urls = doc.getElementsByTagName("dcat:accessURL")
69          for url in urls:
70              u = url.firstChild.data
71              if(u[len(u)-3:]=="csv"):
72                  lurls.append(u)
73
74          for url in lurls:
75              datos_web.descargarArchivosUrl (url, "../data/historico/" +
                    os.path.basename(url) , 2)
76              lista.append("../data/historico/" + os.path.basename(url))
77
78          return lista
79
80      except ValueError as descripcion:
81          print("Ocurrio un error previsto:", descripcion)
82
83
84  def get_tiempo_real ():
85      """Descarga el dataset con la informacion de las medidas realizadas por la
            estaciones (actualizaciones cda hora).
86      Lo guarda en disco.
87
88      Excepciones:
89      ValueError -- Si la funcion descargarArchivosUrl propaga algun error.
90
91      """
92
```

```
93      urlDescargar =
            'https://datos.madrid.es/egob/catalogo/212531-0-calidad-aire-tiempo-
94      real.dcat'
95      urlGuardar = "../data/tiempo_real/212531-0-calidad-aire-tiempo-real.dcat"
96
97      if os.path.exists("../data/tiempo_real/") == False:
98          os.mkdir("../data/tiempo_real/")
99
100     try:
101         datos_web.descargarArchivosUrl (urlDescargar, urlGuardar, 0)
102         doc = minidom.parse(urlGuardar)
103         urls = doc.getElementsByTagName("dcat:accessURL")
104         url = urls[-1].firstChild.data
105         datos_web.descargarArchivosUrl (url[:-3] + "txt", "../data/tiempo_real/"
                + os.path.basename(url)[:-3] + "txt" , 0)
106
107
108     except ValueError as descripcion:
109         print("Ocurrio un error previsto:", descripcion)
110
111
112 def get_climatologia_historico():
113     """Descarga el fichero JSON con los datos historicos de la climatologia de
            Madrid desde el anio 2001
114     hasta la actualidad. Lo guarda en disco.
115
116     Excepciones:
117     ValueError -- Si la funcion descargarArchivosUrl propaga algun error.
118
119     """
120
121     api_key = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJsdXptYXJpYS5tdHRAZ21haWwuY
122     29tIiwianRpIjoiMGRjZWYzNWMtOTExYS00YWI4LTlkY2EtN2M3OGM4ZmUyMDA4IiwiaX
123     NzIjoiQUVNRVQiLCJpYXQiOjE1NjE4MDc0NjgsInVzZXJJZCI6IjBkY2VmMzVjLTkxMWE
```

```
124     tNGFiOC05ZGNhLTdjNzhjOGZlMjAwOCIsInJvbGUiOiIifQ.ZGKWOZvIfIcSxeLgdM4dY
125     k1cZoBGW-v3uyB6k_Rm6zI"
126     url = "https://opendata.aemet.es/opendata/api/valores/climatologicos/
127     diarios/datos/fechaini/2014-06-01T00%3A00%3A00UTC/fechafin/2019-05-31
128     T23%3A59%3A59UTC/estacion/3195"
129     url2 = "https://opendata.aemet.es/opendata/api/valores/climatologicos/
130     diarios/datos/fechaini/2009-06-01T00%3A00%3A00UTC/fechafin/2014-05-31T
131     23%3A59%3A59UTC/estacion/3195"
132     url3 = "https://opendata.aemet.es/opendata/api/valores/climatologicos/
133     diarios/datos/fechaini/2004-06-01T00%3A00%3A00UTC/fechafin/2009-05-31T
134     23%3A59%3A59UTC/estacion/3195"
135     url4 = "https://opendata.aemet.es/opendata/api/valores/climatologicos/
136     diarios/datos/fechaini/2001-01-01T00%3A00%3A00UTC/fechafin/2004-05-31T
137     23%3A59%3A59UTC/estacion/3195"
138
139     if os.path.exists("../data/clima_historico") == False:
140         os.mkdir("../data/clima_historico")
141
142     querystring = {"api_key":api_key}
143
144     headers = {
145         'cache-control': "no-cache"
146         }
147
148     response = requests.request("GET", url, headers=headers, params=querystring)
149     decoded = json.loads(response.content)
150     response2 = requests.request("GET", url2, headers=headers,
            params=querystring)
151     decoded2 = json.loads(response2.content)
152     response3 = requests.request("GET", url3, headers=headers,
            params=querystring)
153     decoded3 = json.loads(response3.content)
154     response4 = requests.request("GET", url4, headers=headers,
            params=querystring)
```

```
155     decoded4 = json.loads(response4.content)
156 
157     try:
158         datos_web.descargarArchivosUrl(decoded["datos"],
                "../data/clima_historico/aemet_historico_datos_1.json", 0)
159         datos_web.descargarArchivosUrl(decoded["metadatos"],
                "../data/clima_historico/aemet_historico_metadatos_1.json", 0)
160         datos_web.descargarArchivosUrl(decoded2["datos"],
                "../data/clima_historico/aemet_historico_datos_2.json", 0)
161         datos_web.descargarArchivosUrl(decoded2["metadatos"],
                "../data/clima_historico/aemet_historico_metadatos_2.json", 0)
162         datos_web.descargarArchivosUrl(decoded3["datos"],
                "../data/clima_historico/aemet_historico_datos_3.json", 0)
163         datos_web.descargarArchivosUrl(decoded3["metadatos"],
                "../data/clima_historico/aemet_historico_metadatos_3.json", 0)
164         datos_web.descargarArchivosUrl(decoded4["datos"],
                "../data/clima_historico/aemet_historico_datos_4.json", 0)
165         datos_web.descargarArchivosUrl(decoded4["metadatos"],
                "../data/clima_historico/aemet_historico_metadatos_4.json", 0)
166     except ValueError as descripcion:
167         print("Ocurrio un error previsto:", descripcion)
168 
169 
170 def get_climatologia_tiempo_real():
171     """Descarga el dataset con la informacion de la climatologia en el dia. Lo
            guarda en disco.
172     El valor de las precipitaciones en mm, se obtiene de la web www.eltiempo.es
            ya que el dataset
173     anterior cuenta con el porcentaje de probabilidad de lluvia, pero no con el
            valor en mm
174 
175     Devuelve el valor de las precipitaciones obtenidas por web scrapping
176 
177     Excepciones:
```

```
178     ValueError -- Si la funcion descargarArchivosUrl propaga algun error.
179
180     """
181
182     api_key = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJsdXptYXJpYS5tdHRAZ21haWwuY
183     29tIiwianRpIjoiMGRjZWYzNWMtOTExYS00YWI4LTlkY2EtN2M3OGM4ZmUyMDA4IiwiaX
184     NzIjoiQUVNRVQiLCJpYXQiOjE1NjE4MDc0NjgsInVzZXJJZCI6IjBkY2VmMzVjLTkxMWE
185     tNGFiOC05ZGNhLTdjNzhjOGZlMjAwOCIsInJvbGUiOiIifQ.ZGKWOZvIfIcSxeLgdM4dY
186     k1cZoBGW-v3uyB6k_Rm6zI"
187     url = "https://opendata.aemet.es/opendata/api/prediccion/especifica/
188     municipio/diaria/28079"
189
190     if os.path.exists("../data/clima_real") == False:
191         os.mkdir("../data/clima_real")
192
193     querystring = {"api_key":api_key}
194
195     headers = {
196         'cache-control': "no-cache"
197         }
198
199     response = requests.request("GET", url, headers=headers, params=querystring)
200     decoded = json.loads(response.content)
201
202     try:
203         datos_web.descargarArchivosUrl(decoded["datos"],
                "../data/clima_real/predicion_diaria.json", 0)
204         datos_web.descargarArchivosUrl(decoded["metadatos"],
                "../data/clima_real/metadatos_predicion_diaria.json", 0)
205
206     except ValueError as descripcion:
207         print("Ocurri\xf3 un error previsto:", descripcion)
208
209     page = requests.get('https://www.eltiempo.es/madrid.html')
```

```
210     tree = html.fromstring(page.content)
211
212     precip = tree.xpath('//*[@id="cityTable"]/div/article/
213     div[1]/div[2]/div[1]/div[7]/span[2]/text()')
214
215     return str(precip)[2:-5]
216
217
218 def tratar_dataset_estaciones_control():
219     """Procesa el dataset de la estaciones de control.
220     Crea una lista con diccionarios donde cada diccionario son los datos de una
            estacion.
221
222     Devuelve un objeto lista con los diccionarios con la informacion sobre las
            estaciones
223
224     """
225
226     doc = xlrd.open_workbook("../data/estaciones_control/
227     212629-0-estaciones-control-aire.xls")
228     estaciones = doc.sheet_by_index(0)
229
230     lestaciones =[]
231
232     for i in range(estaciones.nrows):
233         if (i > 4) and (i < 29):
234             fila = estaciones.row(i)
235             for j in range(len(fila)):
236                 if (j == 1):
237                     numero = int(fila[j].value)
238                     codigo = int(28079000 + numero)
239                 elif (j == 2):
240                     estacion = fila[j].value
241                 elif (j == 3):
```

```
242                 direccion = fila[j].value
243             elif (j == 4):
244                 longitud = fila[j].value
245             elif (j == 5):
246                 latitud = fila[j].value
247             elif (j == 6):
248                 altitud = int(fila[j].value)
249             elif (j == 7):
250                 tipo_estacion = fila[j].value
251             elif (j == 8):
252                 no2 = False
253                 if fila[j].value == "X":
254                     no2 = True
255             elif (j == 9):
256                 so2 = False
257                 if fila[j].value == "X":
258                     so2 = True
259             elif (j == 10):
260                 co = False
261                 if fila[j].value == "X":
262                     co = True
263             elif (j == 11):
264                 pm10 = False
265                 if fila[j].value == "X":
266                     pm10 = True
267             elif (j == 12):
268                 pm2_5 = False
269                 if fila[j].value == "X":
270                     pm2_5 = True
271             elif (j == 13):
272                 o3 = False
273                 if fila[j].value == "X":
274                     o3 = True
275             elif (j == 14):
```

```
276                     btx = False
277                     if fila[j].value == "X":
278                         btx = True
279                 elif (j == 15):
280                     hc = False
281                     if fila[j].value == "X":
282                         hc = True
283                 elif (j == 16):
284                     uv = False
285                     if fila[j].value == "X":
286                         uv = True
287                 elif (j == 17):
288                     vv = False
289                     if fila[j].value == "X":
290                         vv = True
291                 elif (j == 18):
292                     dv = False
293                     if fila[j].value == "X":
294                         dv = True
295                 elif (j == 19):
296                     tmp = False
297                     if fila[j].value == "X":
298                         tmp = True
299                 elif (j == 20):
300                     hr = False
301                     if fila[j].value == "X":
302                         hr = True
303                 elif (j == 21):
304                     prb = False
305                     if fila[j].value == "X":
306                         prb = True
307                 elif (j == 22):
308                     rs = False
309                     if fila[j].value == "X":
```

```
310                     rs = True
311             elif (j == 23):
312                 ll = False
313                 if fila[j].value == "X":
314                     ll = True
315 
316         est = {'codigo' : codigo, 'numero' : numero, 'estacion' : estacion,
                'direccion' : direccion, 'longitud' : longitud, 'latitud' :
                latitud, 'altitud' : altitud, 'tipo_estacion' : tipo_estacion,
                'no2' : no2, 'so2' : so2, 'co' : co, 'pm10' : pm10, 'pm2_5' :
                pm2_5, 'o3' : o3, 'btx' : btx, 'hc' : hc, 'uv' : uv, 'vv' : vv,
                'dv' : dv, 'tmp' : tmp, 'hr' : hr, 'prb' : prb, 'rs' : rs, 'll' :
                ll}
317         lestaciones.append(est)
318 
319     return lestaciones
320 
321 
322 def tratar_dataset_climatologia_historico ():
323     """Procesa el dataset historico de la climatologia.
324     Crea una nuevo fichero llamado clima_hist.csv con los datos relevantes.
325     Deja los registros ordenados por fecha, desde el mas actual al mas antiguo.
326 
327     """
328 
329     cont_ficher = 1
330 
331     if os.path.exists("../data/entrenamiento") == False:
332         os.mkdir("../data/entrenamiento")
333 
334     f=open("../data/clima_historico/clima_hist.csv","w")
335     f.write("anio;mes;dia;temperatura;viento;precipitacion\n")
336 
337     while cont_ficher < 5:
```

```
338         with open("../data/clima_historico/aemet_historico_datos_"+
                str(cont_ficher) +".json") as file:
339             data = json.load(file)
340 
341             for elemento in data:
342 
343                 try:
344                     fecha = elemento['fecha']
345 
346                     try:
347                         temp = elemento['tmed']
348                     except KeyError:
349                         temp = ""
350                     try:
351                         precip = elemento['prec']
352                     except KeyError:
353                         precip = ""
354                     try:
355                         viento = elemento['velmedia']
356                     except KeyError:
357                         viento = ""
358 
359                 except KeyError:
360                     pass
361 
362                 linea = fecha[:4] + ";" + fecha[5:7] + ";" + fecha[8:] + ";" +
                        temp + ";" + viento + ";" + precip
363                 linea = linea.replace(",",".")
364                 f.write(linea + "\n")
365 
366         cont_ficher += 1
367 
368     f.close()
369 
```

```
370     datos = pd.read_csv ("../data/clima_historico/clima_hist.csv",
            delimiter=";")
371     df = pd.DataFrame(datos)
372     df2 = df.sort_values(['anio', 'mes', 'dia'], ascending=[False, False,
            False])
373     os.remove("../data/clima_historico/clima_hist.csv")
374     df2.to_csv("../data/clima_historico/clima_hist.csv", index=False, sep=";")
375
376
377 def tratar_dataset_historicos_training():
378     """Procesa los datasets resultados de tratar los datos historicos. Unifica
            tanto las medidas de las magnitudes como los valores del clima.
379     Genera una carpeta llamada entrenamiento en la que dentro se hayara una
            carpeta con el codigo de cada estacion que a su vez contendra un
380     fichero con los datos historicos completos de esa estacion.
381     Preparados para su posterior procesado como datos de entrenamiento.
382
383     """
384
385     lista_his_estaciones = get_historicos()
386
387     f=open("../data/entrenamiento/fase1.csv","w")
388
389     f.write('ESTACION;MAGNITUD;ANO;MES;DIA;VALOR\n')
390
391     for archivo in lista_his_estaciones:
392         datos = pd.read_csv (archivo, delimiter=";")
393         df = pd.DataFrame(datos)
394
395         size = df.shape
396         for i in range(size[0]):
397             estacion = str(df['PUNTO_MUESTREO'][i])[0:8]
398             magnitud = str(df['PUNTO_MUESTREO'][i])[9:11]
399             if magnitud[1] == "_":
```

```
400             magnitud = magnitud[0]
401         anio = df['ANO'][i]
402         mes = df['MES'][i]
403
404         d1 = df['D01'][i]
405         if df['V01'][i] == "V":
406             f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                    ';' + str(mes) + ';1;' + str(d1) + '\n')
407         d2 = df['D02'][i]
408         if df['V02'][i] == "V":
409             f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                    ';' + str(mes) + ';2;' + str(d2) + '\n')
410         d3 = df['D03'][i]
411         if df['V03'][i] == "V":
412             f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                    ';' + str(mes) + ';3;' + str(d3) + '\n')
413         d4 = df['D04'][i]
414         if df['V04'][i] == "V":
415             f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                    ';' + str(mes) + ';4;' + str(d4) + '\n')
416         d5 = df['D05'][i]
417         if df['V05'][i] == "V":
418             f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                    ';' + str(mes) + ';5;' + str(d5) + '\n')
419         d6 = df['D06'][i]
420         if df['V06'][i] == "V":
421             f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                    ';' + str(mes) + ';6;' + str(d6) + '\n')
422         d7 = df['D07'][i]
423         if df['V07'][i] == "V":
424             f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                    ';' + str(mes) + ';7;' + str(d7) + '\n')
425         d8 = df['D08'][i]
426         if df['V08'][i] == "V":
```

```
427                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';8;' + str(d8) + '\n')
428             d9 = df['D09'][i]
429             if df['V09'][i] == "V":
430                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';9;' + str(d9) + '\n')
431             d10 = df['D10'][i]
432             if df['V10'][i] == "V":
433                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';10;' + str(d10) + '\n')
434             d11 = df['D11'][i]
435             if df['V11'][i] == "V":
436                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';11;' + str(d11) + '\n')
437             d12 = df['D12'][i]
438             if df['V12'][i] == "V":
439                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';12;' + str(d12) + '\n')
440             d13 = df['D13'][i]
441             if df['V13'][i] == "V":
442                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';13;' + str(d13) + '\n')
443             d14 = df['D14'][i]
444             if df['V14'][i] == "V":
445                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';14;' + str(d14) + '\n')
446             d15 = df['D15'][i]
447             if df['V15'][i] == "V":
448                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';15;' + str(d15) + '\n')
449             d16 = df['D16'][i]
450             if df['V16'][i] == "V":
451                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';16;' + str(d16) + '\n')
```

```
452             d17 = df['D17'][i]
453             if df['V17'][i] == "V":
454                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';17;' + str(d17) + '\n')
455             d18 = df['D18'][i]
456             if df['V18'][i] == "V":
457                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';18;' + str(d18) + '\n')
458             d19 = df['D19'][i]
459             if df['V19'][i] == "V":
460                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';19;' + str(d19) + '\n')
461             d20 = df['D20'][i]
462             if df['V20'][i] == "V":
463                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';20;' + str(d20) + '\n')
464             d21 = df['D21'][i]
465             if df['V21'][i] == "V":
466                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';21;' + str(d21) + '\n')
467             d22 = df['D22'][i]
468             if df['V22'][i] == "V":
469                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';22;' + str(d22) + '\n')
470             d23 = df['D23'][i]
471             if df['V23'][i] == "V":
472                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';23;' + str(d23) + '\n')
473             d24 = df['D24'][i]
474             if df['V24'][i] == "V":
475                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';24;' + str(d24) + '\n')
476             d25 = df['D25'][i]
477             if df['V25'][i] == "V":
```

```
478                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';25;' + str(d25) + '\n')
479             d26 = df['D26'][i]
480             if df['V26'][i] == "V":
481                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';26;' + str(d26) + '\n')
482             d27 = df['D27'][i]
483             if df['V27'][i] == "V":
484                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';27;' + str(d27) + '\n')
485             d28 = df['D28'][i]
486             if df['V28'][i] == "V":
487                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';28;' + str(d28) + '\n')
488             d29 = df['D29'][i]
489             if df['V29'][i] == "V":
490                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';29;' + str(d29) + '\n')
491             d30 = df['D30'][i]
492             if df['V30'][i] == "V":
493                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';30;' + str(d30) + '\n')
494             d31 = df['D31'][i]
495             if df['V31'][i] == "V":
496                 f.write(str(estacion) + ';' + str(magnitud) + ';' + str(anio) +
                        ';' + str(mes) + ';31;' + str(d31) + '\n')
497
498     f.close()
499
500
501     datos = pd.read_csv ("../data/entrenamiento/fase1.csv", delimiter=";")
502     df = pd.DataFrame(datos)
503
504
```

```
505     estaciones = datasets.tratar_dataset_estaciones_control()
506
507     #Guardamos un fichero por cada estacion
508     if os.path.exists("../data/entrenamiento/estaciones") == False:
509         os.mkdir("../data/entrenamiento/estaciones")
510
511     for estacion in estaciones:
512         codigo = int(estacion['codigo'])
513         df2 = df[df['ESTACION'] == codigo]
514         df2 = df2.sort_values(['ANO', 'MES', 'DIA', 'MAGNITUD'],
                ascending=[False, False, False, True])
515         df2.to_csv("../data/entrenamiento/estaciones/" + str(codigo) + ".csv",
                index=False, sep =";")
516
517
518
519     #**************************
520
521     datos_clima = pd.read_csv ("../data/clima_historico/clima_hist.csv",
            delimiter=";")
522     df_clima = pd.DataFrame(datos_clima)
523
524     dic_clima = {}
525     size = df_clima.shape
526     for n in range(size[0]):
527         key = str(df_clima['anio'][n]) + "-" + str(df_clima['mes'][n]) + "-" +
                str(df_clima['dia'][n])
528         valor = str(df_clima['temperatura'][n]) + ";" +
                str(df_clima['viento'][n]) + ";" + str(df_clima['precipitacion'][n])
529         dic_clima.update({key:valor})
530
531     #*************************
532
533     for estacion in estaciones:
```

```
534
535         pathw = "../data/entrenamiento/" + str(estacion['codigo'])
536         pathr = "../data/entrenamiento/estaciones/"
537
538         cab = False
539         if os.path.exists(pathw) == False:
540             os.mkdir(pathw)
541         f=open(pathw + "/data_test_" + str(estacion['codigo']) + ".csv","w")
542
543         no2 = "NaN"
544         no2_c = False
545         no2_c_2 = False
546         so2 = "NaN"
547         so2_c = False
548         so2_c_2 = False
549         co = "NaN"
550         co_c = False
551         co_c_2 = False
552         pm10 = "NaN"
553         pm10_c = False
554         pm10_c_2 = False
555         pm2_5 = "NaN"
556         pm2_5_c = False
557         pm2_5_c_2 = False
558         o3 = "NaN"
559         o3_c = False
560         o3_c_2 = False
561
562         cab = False
563
564         datos = pd.read_csv (pathr + str(estacion['codigo']) + '.csv',
                delimiter=";")
565         df = pd.DataFrame(datos)
566
```

```
567         size = df.shape
568 
569 
570 
571         for k in range(size[0]):
572 
573             if k == 0:
574                 anio = str(df['ANO'][k])
575                 mes = str(df['MES'][k])
576                 dia = str(df['DIA'][k])
577 
578             if dia == str(df['DIA'][k]):
579 
580                 magnitud = str(df['MAGNITUD'][k])
581                 valor = str(df['VALOR'][k])
582 
583                 if magnitud == "1":
584                     so2 = valor
585                     so2_c = True
586                 elif magnitud == "6":
587                     co = valor
588                     co_c = True
589                 elif magnitud == "8":
590                     no2 = valor
591                     no2_c = True
592                 elif magnitud == "9":
593                     pm10 = valor
594                     pm10_c = True
595                 elif magnitud == "10":
596                     pm2_5 = valor
597                     pm2_5_c = True
598                 elif magnitud == "14":
599                     o3 = valor
600                     o3_c = True
```

```
601
602         else:
603             #escribimos la cabecera primero
604             if cab == False:
605                 cabecera = ""
606                 if so2_c == True:
607                     cabecera = cabecera + 'so2;'
608                     so2_c_2 = True
609                 if co_c == True:
610                     cabecera = cabecera + 'co;'
611                     co_c_2 = True
612                 if no2_c == True:
613                     cabecera = cabecera + 'no2;'
614                     no2_c_2 = True
615                 if pm10_c == True:
616                     cabecera = cabecera + 'pm10;'
617                     pm10_c_2 = True
618                 if pm2_5_c == True:
619                     cabecera = cabecera + 'pm2_5;'
620                     pm2_5_c_2 = True
621                 if o3_c == True:
622                     cabecera = cabecera + 'o3;'
623                     o3_c_2 = True
624                 cabecera = cabecera + "temp;viento;precipitacion"
625                 f.write(cabecera + "\n")
626                 cab = True
627
628             #escribimos la linea
629             linea = ""
630             if so2_c_2 == True:
631                 linea = linea + str(so2) + ';'
632             if co_c_2 == True:
633                 linea = linea + str(co) + ';'
634             if no2_c_2 == True:
```

```
635                 linea = linea + str(no2) + ';'
636             if pm10_c_2 == True:
637                 linea = linea + str(pm10) + ';'
638             if pm2_5_c_2 == True:
639                 linea = linea + str(pm2_5) + ';'
640             if o3_c_2 == True:
641                 linea = linea + str(o3) + ';'
642 
643             clave = anio + "-" + mes + "-" + dia
644 
645             if clave in dic_clima:
646                 linea = linea + dic_clima[clave]
647                 f.write(linea + "\n")
648 
649             anio = str(df['ANO'][k])
650             mes = str(df['MES'][k])
651             dia = str(df['DIA'][k])
652 
653             magnitud = str(df['MAGNITUD'][k])
654             valor = str(df['VALOR'][k])
655 
656             if magnitud == "1":
657                 so2 = valor
658             elif magnitud == "6":
659                 co = valor
660             elif magnitud == "8":
661                 no2 = valor
662             elif magnitud == "9":
663                 pm10 = valor
664             elif magnitud == "10":
665                 pm2_5 = valor
666             elif magnitud == "14":
667                 o3 = valor
668 
```

```
669         f.close()
670 
671 
672     os.remove("../data/entrenamiento/fase1.csv")
673     shutil.rmtree("../data/entrenamiento/estaciones")
674 
675     for estacion in estaciones:
676         codigo = int(estacion['codigo'])
677         path = "../data/entrenamiento/" + str(codigo) + "/"
678 
679         datoss = pd.read_csv (path + "data_test_" + str(codigo) + ".csv",
                delimiter=";")
680         df = pd.DataFrame(datoss)
681 
682 
683         f=open(path + "/data_totest_" + str(estacion['codigo']) + ".csv","w")
684         cabeceras = df.dtypes.index.tolist()
685         cab = ""
686         for cabecera in cabeceras:
687             cab = cab + cabecera + ";"
688         cab = cab + "n_predicion"
689         f.write(cab + "\n")
690 
691         size = df.shape
692         for k in range(size[0]):
693             if k < 3:
694                 linea=""
695                 for i in range(size[1]):
696                     linea = linea + str(df[cabeceras[i]][k]) + ";"
697                 prediccion = 'NaN'
698                 linea = linea + str(prediccion)
699                 f.write(linea + "\n")
700 
701             if k >= 4:
```

```
702                 linea=""
703                 for i in range(size[1]):
704                     linea = linea + str(df[cabeceras[i]][k]) + ";"
705                 v_umbral = df['no2'][k-3]
706                 if v_umbral >= 50:
707                     prediccion = 1
708                 elif v_umbral >= 100:
709                     prediccion = 2
710                 elif v_umbral >= 150:
711                     prediccion = 3
712                 elif v_umbral >= 200:
713                     prediccion = 4
714                 else:
715                     prediccion = 0
716                 linea = linea + str(prediccion)
717                 linea = linea.replace('nan','NaN')
718                 linea = linea.replace('Ip','NaN')
719                 f.write(linea + "\n")
720 
721         f.close()
722 
723         os.remove(path + "data_test_" + str(codigo) + ".csv")
724 
725 
726 def crear_input_clasificadores_dataset ():
727     """Procesado final en el que se eliminan los registros nulos y se da
            formato a los datos por cada estacion.
728     Partiendo del archivo de cada estacion data_totest_<estacion>.csv se
            formatearan los datos y se guardaran en dos ficheros,
729     uno para el arbol y el otro para la red neuronal (data_tree_<estacion>.txt
            y data_neuronal_network_<estaccion>.txt)
730 
731     """
732 
```

```
733     estaciones = datasets.tratar_dataset_estaciones_control()
734     for estacion in estaciones:
735         codigo = int(estacion['codigo'])
736 
737         path = "../data/entrenamiento/" + str(codigo) + "/"
738 
739         datos = pd.read_csv (path + "data_totest_" + str(codigo) + ".csv",
                delimiter=";")
740         df = pd.DataFrame(datos)
741         #Eliminamos valores nulos
742         df = df.dropna()
743 
744         cabeceras = df.dtypes.index.tolist()
745         #reseteamos los indices
746         df = df.reset_index(drop = True)
747         #obtenemos el tamanio del df
748         size = df.shape
749 
750 
751         lineaAtributos = ""
752         lineaEtiquetas = ""
753 
754         for k in range(size[0]):
755 
756             lineaEtiquetas = lineaEtiquetas + str(df['n_predicion'][k]) + ","
757             for i in range(len(cabeceras)-1):
758                 cab = cabeceras[i]
759                 valor = df[cab][k]
760                 lineaAtributos = lineaAtributos + str(valor) + ","
761 
762             lineaAtributos = lineaAtributos[:-1] + ",,"
763 
764         lineaEtiquetas = lineaEtiquetas[:-1]
765         lineaAtributos = lineaAtributos[:-2]
```

```
766
767
768         f=open(path + "data_tree_" + str(codigo) + ".txt","w")
769         f.write(str(cabeceras) + "\n")
770         f.write(lineaEtiquetas + "\n")
771         f.write(lineaAtributos + "\n")
772         f.close()
773
774         f=open(path + "data_neural_network_" + str(codigo) + ".txt","w")
775         f.write(str(cabeceras) + "\n")
776         f.write(lineaEtiquetas + "\n")
777         f.write(lineaAtributos + "\n")
778         f.close()
779
780
781 def tratar_dataset_climatologia_tiempo_real():
782     """Tras invocar a la funcion get_climatologia_tiempo_real() para la
            descarga del dataset con las medidas en tiempo real de la climatologia,
            lo procesa.
783
784     Devuelve tres valores, la temperatura, el viento y las precipitaciones.
785
786     """
787
788
789     precip = float(datasets.get_climatologia_tiempo_real())
790
791
792     with open("../data/clima_real/predicion_diaria.json") as file:
793         data = json.load(file)
794
795         #viento
796         contador = 0
797         viento = 0
```

```
798         for elemento in data[0]['prediccion']['dia'][0]['viento']:
799             viento = viento + int(elemento['velocidad'])
800             contador += 1
801         viento = round(viento / contador, 5)
802 
803         #temperatura
804         contador = 0
805         temp = 0
806         for elemento in data[0]['prediccion']['dia'][0]['temperatura']['dato']:
807             temp = temp + elemento['value']
808             if int(elemento['value']) != 0:
809                 contador += 1
810         temp = round(temp / contador, 5)
811 
812         return (temp, viento, precip)
813 
814 
815 def tratar_dataset_tiempo_real():
816     """Tras invocar a la funcion get_tiempo_real() para la descarga del dataset
            con las medidas en tiempo real de las estaciones, lo procesa.
817     Genera un nuevo fichero unico con las medidas de todas las estaciones.
818 
819     Devuelve un string con la url donde se encuentra almacenado el fichero
            salida
820 
821     """
822 
823     get_tiempo_real()
824 
825     fw=open("../data/tiempo_real/calidad_aire_real_procesado.csv","w")
826     fw.write('ESTACION;MAGNITUD;FECHA;HORA;VALOR\n')
827 
828     fr=open("../data/tiempo_real/212531-7916318
829     -calidad-aire-tiempo-real.txt")
```

```
830     for linea in fr:
831         datos = linea.split(',')
832         magnitud=""
833         if int(datos[3]) == 1:
834             magnitud = 'so2'
835         elif int(datos[3]) == 6:
836             magnitud = 'co'
837         elif int(datos[3]) == 8:
838             magnitud = 'no2'
839         elif int(datos[3]) == 9:
840             magnitud = 'pm10'
841         elif int(datos[3]) == 10:
842             magnitud = 'pm2_5'
843         elif int(datos[3]) == 14:
844             magnitud = 'o3'
845 
846         ahora = datetime.now()
847         hora = ahora.hour
848         if ahora.minute < 35:
849             if hora == 0:
850                 hora = 22
851             elif hora == 1:
852                 hora = 23
853             else:
854                 hora = hora - 2
855         else:
856             if hora == 0:
857                 hora = 23
858             else:
859                 hora = hora -1
860 
861 
862         valor = str(datos[int(hora)*2+9])
863         if magnitud != "":
```

```
864             fw.write(str(datos[0]) + str(datos[1]) + str(datos[2]) + ";" +
                    magnitud + ";" + str(datos[8]) + "/" + str(datos[7]) + "/" +
                    str(datos[6]) + ";" + str(hora) + ":00" + ";" + str(valor) + "\n")
865
866     fr.close()
867     fw.close()
868
869     return "../data/tiempo_real/calidad_aire_real_procesado.csv"
```

Fragmento de código 27: datasets.py del paquete gestion_datos

datos_web.py (Fragmento de código 28)

```
1
2  '''
3  Created on 12 may. 2019
4
5  @author: Luz Maria Martinez
6  '''
7
8  import wget
9  from unipath import Path
10 import os
11
12 def descargarArchivosUrl (urlDescargar = None, urlGuardar = None, accion = 0):
13     """Descarga un archivo de la web mediante la url y lo guarda en disco.
14
15     Devuelve True si se ha descargado y guardado el archivo correctamente, sino
           devolvera False.
16
17     Parametros:
18     urlDescargar -- url del archivo a descargar
19     urlGuardar -- ruta donde se desea guardar el dataset (incluido el nombre
           del fichero)
20     accion -- 0 = reemplazar el archivo si existe, 1 = guardar con otro nombre
           sin reemplazar (nombre (n), n=1,2,3...), 2 = no sustituir ni guardar el
           archivo si existe.
21
22     Excepciones:
23     ValueError -- Si (urlDescargar = None) o (urlGuardar = None)
24
25     """
26
27     if (urlDescargar == None) or (urlGuardar == None):
28         raise ValueError('Error en las rutas')
29     else:
```

```
30        f = Path(urlGuardar)
31        existe = f.exists()
32
33        if (accion == 2) and (existe == True):
34            return False
35        else:
36            if (accion == 0) and (existe == True):
37                os.remove(urlGuardar)
38            wget.download(urlDescargar, urlGuardar)
39            return True
```

Fragmento de código 28: datos_web.py del paquete gestion_datos

Paquete ia

trees.py (Fragmento de código 29)

```
1
2  '''
3  Created on 10 jun. 2019
4
5  @author: Luz Maria Martinez
6  '''
7
8  from sklearn import tree
9  from sklearn.tree.export import export_text
10 from sklearn.model_selection import train_test_split
11 from sklearn.tree import export_graphviz
12 from sklearn.preprocessing import StandardScaler
13 from sklearn.metrics import confusion_matrix, classification_report
14 import numpy as np
15
16
17
18 def generar_arbol (codigo, profundidad=5, estadisticas=True):
19     """Genera un arbol de decision (clasificador)
20
21     Devuelve el arbol generado y el porcentaje de acierto al validar el modelo
           con el conjunto de entrenamiento.
22
23     Parametros:
24     codigo -- codigo de la estacion de calidad del aire
25     profundidad -- profundidad maxima del arbol. Por defecto 5
26     estadisticas -- True si se quiere exportar un fichero con estadisticas.
           False si no se quiere generar. Por defecto True
27
28
29     """
```

```
30
31      path = "../data/entrenamiento/" + str(codigo) + "/"
32
33      f = open(path + "data_tree_" + str(codigo) + ".txt")
34      cab = str(f.readline())[:-1]
35      cab = cab.replace("'","")
36      cab = cab.replace(" ","")
37      cabeceras = cab[1:-1].split(",")
38      l_etiquetas = str(f.readline())[:-1].split(",")
39      atributos = str(f.readline())[:-1].split(",,")
40      f.close()
41
42      l_atributos = []
43      for atributo in atributos:
44          l_atributos.append(atributo.split(","))
45
46      l_etiquetas = np.array(l_etiquetas)
47      l_atributos = np.array(l_atributos)
48
49      etiquetas = set(l_etiquetas)
50      clases = sorted(list(etiquetas))
51
52
53      #Separamos los datos que vamos a utilizar para entrenar y para validar.
54      #datos de validacion (0.25) / datos de entrenamiento (0.75)
55      X_train, X_test, y_train, y_test =
            train_test_split(l_atributos,l_etiquetas,test_size = 0.25,random_state =
            0)
56
57      #Normalizacion de los datos
58      sc = StandardScaler()
59      X_train = sc.fit_transform(X_train)
60      X_test = sc.transform(X_test)
61
```

```
62    #Creamos el arbol con la profundidad indicada
63    arbol = tree.DecisionTreeClassifier(max_depth=profundidad, criterion =
          'entropy', random_state = 0)
64
65
66    #Entrenamos el arbol
67    arbol.fit(X_train, y_train)
68
69    #Exportar el arbol en texto
70    r = export_text(arbol, feature_names=cabeceras[:-1])
71    f=open(path + "export_tree_text_" + str(codigo) + ".txt","w")
72    f.write(r)
73    f.close()
74
75    #Exportar datos del arbol .dot
76    export_graphviz(arbol,out_file=path + 'export_tree_' + str(codigo) +
          '.dot',class_names=clases,
77                feature_names=cabeceras[:-1],impurity=False,filled=True)
78
79
80
81    # Predicion de los resultados del bloque test
82    y_pred = arbol.predict(X_test)
83    #Matriz de confusion
84    cm = confusion_matrix(y_test, y_pred)
85
86    #Reportes
87    report = classification_report(y_test, y_pred)
88
89
90    if estadisticas == True:
91        f=open(path + "export_tree_statistics_" + str(codigo) + ".txt","w")
92        f.write("************************ CLASS ************************\n")
93        for i in range(len(clases)):
```

```
94            f.write(str(clases[i]) + "\n")
95        f.write("\n\n")
96        f.write("********************* MAX DEPTH ************************\n")
97        f.write(str(profundidad) + "\n")
98        f.write("\n\n")
99        f.write("***************** FEATURE IMPORTANCES *****************\n")
100       featur_imp = arbol.feature_importances_
101       for i in range(len(cabeceras[:-1])):
102           f.write(str(cabeceras[i]) + ": " + str(featur_imp[i]) + "\n")
103       f.write("\n\n")
104       f.write("************************ SCORE ************************\n")
105       f.write("With Test data: " + str(arbol.score(X_test, y_test)) + "\n")
106       f.write("With Training data: " + str(arbol.score(X_train, y_train)) +
              "\n")
107       f.write("\n\n")
108       f.write("****************** CONFUSION MATRIX ********************\n")
109       if len(clases)==2:
110           f.write("\t0\t1\n")
111           f.write("---------------------\n")
112           f.write("0 |\t"+ str(cm[0][0]) +"\t"+ str(cm[0][1]) +"\n")
113           f.write("1 |\t"+ str(cm[1][0]) +"\t"+ str(cm[1][1]) +"\n")
114       if len(clases)==3:
115           f.write("\t0\t1\t2\n")
116           f.write("-------------------------------\n")
117           f.write("0 |\t"+ str(cm[0][0]) +"\t"+ str(cm[0][1]) +"\t"+
                  str(cm[0][2]) +"\n")
118           f.write("1 |\t"+ str(cm[1][0]) +"\t"+ str(cm[1][1]) +"\t"+
                  str(cm[1][2]) +"\n")
119           f.write("2 |\t"+ str(cm[2][0]) +"\t"+ str(cm[2][1]) +"\t"+
                  str(cm[2][2]) +"\n")
120       if len(clases)==4:
121           f.write("\t0\t1\t2\t3\n")
122           f.write("-----------------------------------------------\n")
123           f.write("0 |\t"+ str(cm[0][0]) +"\t"+ str(cm[0][1]) +"\t"+
```

```
                str(cm[0][2]) +"\t"+ str(cm[0][3]) +"\n")
124             f.write("1 |\t"+ str(cm[1][0]) +"\t"+ str(cm[1][1]) +"\t"+
                str(cm[1][2]) +"\t"+ str(cm[1][3]) +"\n")
125             f.write("2 |\t"+ str(cm[2][0]) +"\t"+ str(cm[2][1]) +"\t"+
                str(cm[2][2]) +"\t"+ str(cm[2][3]) +"\n")
126             f.write("3 |\t"+ str(cm[3][0]) +"\t"+ str(cm[3][1]) +"\t"+
                str(cm[3][2]) +"\t"+ str(cm[3][3]) +"\n")
127         f.write("\n\n")
128         f.write("************************ REPORT **********************\n")
129         f.write(report)
130         f.close()
131
132     print(str(codigo) + ": tree created")
133     return (arbol, arbol.score(X_test, y_test))
```

Fragmento de código 29: trees.py del paquete ia

neural_networks.py (Fragmento de código 30)

```
1
2  '''
3  Created on 10 jun. 2019
4
5  @author: Luz Maria Martinez
6  '''
7
8  import numpy as np
9  from sklearn.model_selection import train_test_split
10 from sklearn.metrics import confusion_matrix, classification_report
11 from sklearn.neural_network import MLPClassifier
12
13 def generar_red_neuronal (codigo, estadisticas=True):
14     """Genera una red neuronal (clasificadora)
15
16     Devuelve la red neuronal generada y el porcentaje de acierto al validar el
           modelo con el conjunto de entrenamiento.
17
18     Parametros:
19     codigo -- codigo de la estacion de calidad del aire
20     estadisticas -- True si se quiere exportar un fichero con estadisticas.
           False si no se quiere generar. Por defecto True
21
22
23     """
24
25     path = "../data/entrenamiento/" + str(codigo) + "/"
26
27     f = open(path + "data_neural_network_" + str(codigo) + ".txt")
28     cab = str(f.readline())[:-1]
29     cab = cab.replace("'","")
30     cab = cab.replace(" ","")
31     cabeceras = cab[1:-1].split(",")
```

```
32    l_etiquetas = str(f.readline())[:-1].split(",")
33    atributos = str(f.readline())[:-1].split(",,")
34    f.close()
35
36    l_atributos = []
37    for atributo in atributos:
38        l_atributos.append(atributo.split(","))
39
40    etiquetas = set(l_etiquetas)
41    clases = sorted(list(etiquetas))
42
43    l_atributos = np.array(l_atributos, "float32")
44    l_etiquetas = np.array(l_etiquetas, "float32")
45
46
47    #Separamos los datos que vamos a utilizar de entrenamiento y para realizar
          el test
48    #datos de pruebas (0.25) de los de entrenamiento (0.75)
49    X_train, X_test, y_train, y_test =
          train_test_split(l_atributos,l_etiquetas,test_size = 0.25,random_state =
          0)
50
51    #Creamos la red neuronal
52    red = MLPClassifier(max_iter=100000, hidden_layer_sizes=(50,25))
53    #Entrenamos la red neuronal
54    red.fit(X_train, y_train)
55
56    #Predicion de los resultados del bloque test
57    y_pred = red.predict(X_test)
58    #Matriz de confusion
59    cm = confusion_matrix(y_test, y_pred)
60    #Reportes
61    report = classification_report(y_test, y_pred)
62
```

```
63
64
65      if estadisticas == True:
66          f=open(path + "export_neural_network_statistics_" + str(codigo) +
                ".txt","w")
67          f.write("************************ CLASS ************************\n")
68          for i in range(len(clases)):
69              f.write(str(clases[i]) + "\n")
70          f.write("\n\n")
71          f.write("********************** FEATURES **********************\n")
72          for i in range(len(cabeceras[:-1])):
73              f.write(str(cabeceras[i]) + "\n")
74          f.write("\n\n")
75          f.write("************************ SCORE ************************\n")
76          f.write("With Test data: " + str(red.score(X_test, y_test)) + "\n")
77          f.write("With Training data: " + str(red.score(X_train, y_train)) + "\n")
78          f.write("\n\n")
79          f.write("****************** CONFUSION MATRIX ******************\n")
80          if len(clases)==2:
81              f.write("\t0\t1\n")
82              f.write("---------------------\n")
83              f.write("0 |\t"+ str(cm[0][0]) +"\t"+ str(cm[0][1]) +"\n")
84              f.write("1 |\t"+ str(cm[1][0]) +"\t"+ str(cm[1][1]) +"\n")
85          if len(clases)==3:
86              f.write("\t0\t1\t2\n")
87              f.write("--------------------------------\n")
88              f.write("0 |\t"+ str(cm[0][0]) +"\t"+ str(cm[0][1]) +"\t"+
                    str(cm[0][2]) +"\n")
89              f.write("1 |\t"+ str(cm[1][0]) +"\t"+ str(cm[1][1]) +"\t"+
                    str(cm[1][2]) +"\n")
90              f.write("2 |\t"+ str(cm[2][0]) +"\t"+ str(cm[2][1]) +"\t"+
                    str(cm[2][2]) +"\n")
91          if len(clases)==4:
92              f.write("\t0\t1\t2\t3\n")
```

```
93            f.write("-----------------------------------------------\n")
94            f.write("0 |\t"+ str(cm[0][0]) +"\t"+ str(cm[0][1]) +"\t"+
                  str(cm[0][2]) +"\t"+ str(cm[0][3]) +"\n")
95            f.write("1 |\t"+ str(cm[1][0]) +"\t"+ str(cm[1][1]) +"\t"+
                  str(cm[1][2]) +"\t"+ str(cm[1][3]) +"\n")
96            f.write("2 |\t"+ str(cm[2][0]) +"\t"+ str(cm[2][1]) +"\t"+
                  str(cm[2][2]) +"\t"+ str(cm[2][3]) +"\n")
97            f.write("3 |\t"+ str(cm[3][0]) +"\t"+ str(cm[3][1]) +"\t"+
                  str(cm[3][2]) +"\t"+ str(cm[3][3]) +"\n")
98        f.write("\n\n")
99        f.write("********************** REPORT **********************\n")
100       f.write(report)
101       f.close()
102
103   print(str(codigo) + ": neural network created")
104   return (red, red.score(X_test, y_test))
```

Fragmento de código 30: neural_networks.py del paquete ia