Predicción a corto plazo de la energía generada por una placa solar

Pablo Aragón Moreno
María Castañeda López
Abel Coronado López

GRADO EN INGENIERÍA INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Grado en Ingeniería Informática

16 de Junio de 2017

Director/es y/o colaborador:

José Ignacio Gómez
Christian Tenllado

Autorización de difusión

Pablo Aragón Moreno
María Castañeda López
Abel Coronado López

16 de Junio de 2017

El/la abajo firmante, matriculado/a en el Grado en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado: "Predicción a corto plazo de la energía generada por una placa solar", realizado durante el curso académico 2016-2017 bajo la dirección de Jose Ignacio Gómez, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Pablo Aragón Moreno

María Castañeda López

Abel Coronado López

Director: Jose Ignacio Gómez

Director: Christian Tenllado

Resumen en castellano

Proyecto que intenta predecir la radiación solar que habrá en un punto en un intervalo de tiempo definido utilizando modelos predictivos.

Palabras clave

Radiación solar

Placa solar

Raspberry Pi

Piranómetro

Inteligencia artifical

Algoritmos

Predicción

Abstract

Project that attempts to predict solar radiation at a point in a defined time interval using predictive models.

Keywords

Solar radiation

Solar panel

Raspberry Pi

Pyranometer

Machine learning

Algorithms

Prediction

Índice general

Índice I

Agradecimientos IV

1. Introducción 1
- 1.1. Motivación . . . 2
- 1.2. Breve descripción del sistema . . . 3
  - 1.2.1. Nodo . . . 3
  - 1.2.2. Servidor de datos . . . 3
  - 1.2.3. Servidor de predicción . . . 4
  - 1.2.4. Sistema de visualización . . . 4

2. Introduction 5
- 2.1. Motivation . . . 6
- 2.2. Brief description of the system . . . 7
  - 2.2.1. Node . . . 7
  - 2.2.2. Data server . . . 7
  - 2.2.3. Prediction server . . . 7
  - 2.2.4. Display system . . . 7

3. Nodo 8
- 3.1. Introducción . . . 8
- 3.2. Elementos hardware utilizados . . . 8
  - 3.2.1. Raspberry Pi 3 . . . 8
  - 3.2.2. Sensor de temperatura y humedad . . . 9
  - 3.2.3. Piranómetro y ADC . . . 11
- 3.3. Componentes software y librerías externas . . . 12
  - 3.3.1. Librerías externas . . . 13
  - 3.3.2. Comunicación . . . 14

4. Servidor de datos 15
- 4.1. Introducción . . . 15
- 4.2. MQTT . . . 16
  - 4.2.1. Seguridad . . . 17
  - 4.2.2. QoS . . . 17

5. Exploración del modelo predictivo **19**
5.1. Introducción a los modelos predictivos . . . 19
5.2. Conjunto de datos . . . 21
5.3. Algoritmos utilizados . . . 24
5.3.1. Regresión Lineal . . . 24
5.3.2. SVR (Support Vector Regression) . . . 25
5.3.3. Redes Neuronales . . . 26
5.4. Método de estudio . . . 28
5.4.1. Grid Search . . . 28
5.4.2. Sistema de entrenamiento . . . 30
5.5. Descripción de las librerías usadas . . . 32
5.5.1. Scikit-learn . . . 32
5.5.2. NumPy . . . 32

6. Predicción y visualización **33**
6.1. Servidor de predicción . . . 33
6.2. Sistema de visualización . . . 34

7. Resultados **36**
7.1. Regresión Lineal . . . 37
7.2. SVR . . . 39
7.3. Redes Neuronales . . . 41

8. Instalación y puesta en marcha **44**
8.1. Nodo . . . 44
8.1.1. Requisitos . . . 44
8.1.2. Ficheros . . . 45
8.1.3. Dependencias . . . 46
8.1.4. Puesta en marcha del nodo . . . 47
8.2. Servidor de datos . . . 47
8.2.1. Requisitos previos . . . 47
8.3. Servidor de predicción . . . 48
8.3.1. Requisitos previos . . . 48
8.3.2. Instalación . . . 48

9. Conclusiones **49**
9.1. Estado del proyecto . . . 49
9.2. Posibles ampliaciones . . . 50

10. Conclusions **51**
10.1. Project status . . . 51
10.2. Possible extensions . . . 52

11.Trabajo del Alumno **53**
11.1. Pablo Aragón Moreno 53
11.1.1. Investigación 53
11.1.2. Desarrollo 54
11.1.3. Documentación 54
11.2. María Castañeda López 55
11.2.1. Investigación 55
11.2.2. Desarrollo 56
11.2.3. Documentación 56
11.3. Abel Coronado López 57
11.3.1. Investigación 57
11.3.2. Desarrollo 58
11.3.3. Documentación 59

Bibliografía **60**

12.Anexo: Gráficas Regresión Lineal **61**

13.Anexo: Gráficas SVR **63**

14.Anexo: Gráficas Redes Neuronales **65**

15.Anexo: Calidad del código **67**

Agradecimientos

A nuestros directores del proyecto Jose Ignacio Gómez y Christian Tenllado por su implicación, atención y paciencia. Siempre dispuestos a orientarnos a lo largo de todo el proyecto y de enseñarnos aquello que no conocíamos.

Capítulo 1

Introducción

Este proyecto tiene como objetivo predecir, a través de herramientas informáticas, la irradiación solar en un punto teniendo en cuenta factores geográficos y meteorológicos.

La irradiación solar se puede usar como entrada de modelos físicos para la predicción de la energía. De esta manera podremos saber lo que podría generar una placa solar en un momento dado.

Ya existen modelos teóricos para saber cuál va a ser la irradiación solar en un lugar, a partir de la inclinación de los rayos solares en función de la longitud, la latitud, el día del año, la hora del día y suponiendo un cielo en ausencia de nubes. Estos modelos no tienen en cuenta los factores climáticos que tienen un gran peso en la disipación de la radiación.

En la figura 1.1 podemos observar una gráfica en la que aparecen los resultados de distintos modelos teóricos de predicción de la radiación en el verano de 2015. La gráfica representa la radiación media de todo el verano durante las horas del día.

Aunque se observa en la gráfica que la predicción se acerca a la realidas, la radiación solar no solo se ve afectada por los parámetros de estos modelos. También varía debido a diversos factores de las variaciones locales en la atmósfera como por ejemplo el vapor de agua, las nubes o la contaminación. Este proyecto quiere acercar todavía más esta predicción haciendo uso de algunos de estos factores.

Para llevar a cabo esta predicción en este proyecto se han utilizado como variables atmosféricas la humedad, la temperatura y la propia radiación solar. La manera de resolver

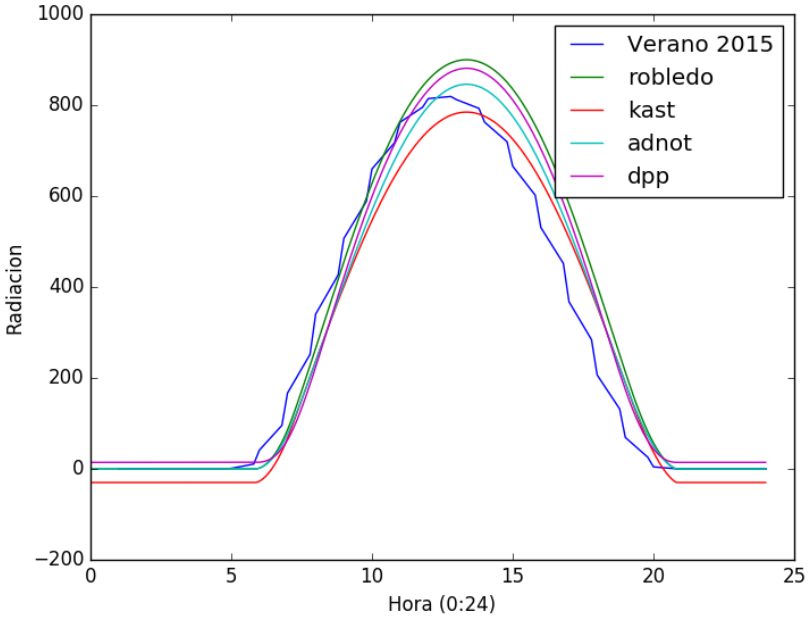

Figura 1.1: *Modelo verano 2015.*

este problema han sido algoritmos de "Machine Learning" (explicados en el capítulo 5).

1.1. Motivación

Hoy en día es cada vez más importante el uso de las energías renovables y así, utilizar cada vez menos los combustibles fósiles. Esto es debido a que estas no producen gases de efecto invernadero que son los causantes del cambio climático, tampoco producen emisiones contaminantes, son inagotables y generan residuos fáciles de tratar.

También tienen algunos inconvenientes como, por ejemplo, el gran impacto visual que tienen y las grandes cantidades de terreno que se necesitan para poder conseguir una cantidad significante de energía. Además no siempre se obtiene la misma cantidad de energía. Depende, por ejemplo, de la cantidad de sol o de viento en ese momento.

Este último inconveniente es el que hace que las compañías productoras y distribuidoras sean reacias al uso de estas energías.

Debido a la liberalización del sector energético, la compra-venta de la energía se realiza en subastas cada hora. Las compañías, deben conocer su capacidad de producción para poder adecuarla a la demanda de estas subastas. Debido a la incertidumbre de cuánta energía se va a producir, es difícil desarrollar estrategias de compra de energía a otros productores y adecuación de su producción y capacidad de reserva.

En energías renovables como la eólica, los modelos predictivos disponibles actualmente son bastante precisos y su impulso es más fácil que otras como la energía solar la cual es más dificil de predecir.

Se podría decir que todavía no existen modelos realmente buenos para predecir la energía solar. Hay algunas empresas como Aleasoft, que realizan estudios predictivos obteniendo previsiones horarias en tiempo real desde un día hasta diez días, pero sería necesario conocer la producción con un intervalo de una o dos horas.

Para ello, este proyecto intenta predecir a corto plazo la radiación solar y poder aplicar esta en una instalación de granja solar.

1.2. Breve descripción del sistema

Este sistema cuenta con cuatro grandes módulos de trabajo para llevar a cabo su función: nodo, servidor de datos, servidor de resultados y visualizador de datos. Ver figura 1.2.

1.2.1. Nodo

Encargado de recoger los datos necesarios, está formado por distintos sensores que recogen la información meteorológica necesaria y una pequeña placa programable encargada de enviar esta información al servidor de datos.

1.2.2. Servidor de datos

El servidor de datos, es el encargado de recibir, almacenar y distribuir la información obtenida por el nodo.

1.2.3. Servidor de predicción

Tiene como función recoger la información almacenada del servidor de datos, procesarla para obtener la predicción y enviarla al visualizador de datos. Puede estar, o no, en la misma máquina que el servidor de datos.

1.2.4. Sistema de visualización

Muestra los resultados en forma de gráficas para facilitar su análisis.

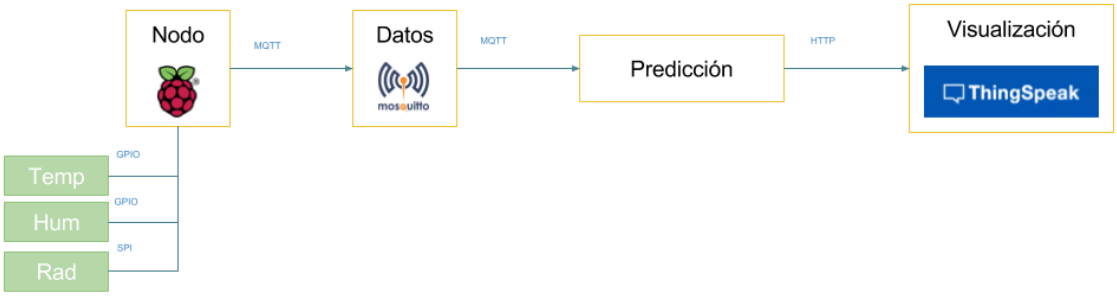

Figura 1.2: *Diagrama de los componentes del sistema con sus protocolos de comunicación.*

Capítulo 2

Introduction

This project aims to predict, through computer tools, the solar radiation at one point taking into account geographic and meteorological factors.

There are already theoretical models to know how much solar irradiation will be in a place, from the inclination of the sun rays in the function of the length, the latitude, the day of the year, the time of day and assuming a sky in the absence of clouds. These models do not take into account the climatic factors that have a great weight in the dissipation of the radiation.

In the picture 1.1 we can observe a graph showing the results of different theoretical models of radiation prediction in the summer of 2015. The graph represents the average radiation of the whole summer during the hours of the day.

Although it is observed in the graph that the prediction approaches the reality, the solar radiation is not only affected by the parameters of these models. It also varies due to various factors of local variations in the atmosphere such as water vapor, clouds or pollution. This project wants to bring this prediction even closer by making use of some of these factors.

To carry out this prediction in this project, the humidity, temperature and the solar radiation itself have been used as atmospheric variables. The way to solve this problem have been algorithms of " Machine Learning " (explained in chapter 5).

2.1. Motivation

Nowadays, the use of renewable energies is becoming more and more important and, therefore, to use less and less fossil fuels. This is because they do not produce greenhouse gases that are the cause of climate change, they do not produce pollutant emissions, they are inexhaustible and generate waste easy to treat.

They also have some drawbacks, such as the great visual impact they have and the large amounts of land needed to get a significant amount of energy. In addition you do not always get the same amount of energy. It depends, for example, on the amount of sun or wind at that time.

This last drawback is that makes the producing and distributing companies are reluctant to use these energies.

Due to the liberalization of the energy sector, energy buying and selling takes place at hourly auctions. The companies must know their production capacity in order to adapt it to the demand of these auctions. Due to the uncertainty of how much energy is going to be produced, it is difficult to develop strategies to purchase energy from other producers and to adjust their production and reserve capacity.

In renewable energies such as wind, currently available predictive models are quite accurate and their momentum is easier than others such as solar energy which is more difficult to predict.

It could be said that there are still no really good models to predict solar energy. There are some companies like Aleasoft, that perform predictive studies obtaining real-time forecasts from one day to ten days, but it would be necessary to know the production with an interval of one or two hours.

For this reason, this project tries to predict in the short term the solar radiation and to be able to apply this in a solar farm installation.

2.2. Brief description of the system

This system has four large work modules to perform its function: node, data server, results server and data viewer. View picture 1.2.

2.2.1. Node

In charge of collecting the necessary data, it is formed by different sensors that collect the necessary meteorological information and a small programmable board in charge of sending this information to the data server.

2.2.2. Data server

The data server is responsible for receiving, storing and distributing the information obtained by the node.

2.2.3. Prediction server

It has as function to collect the stored information of the data server, to process it to obtain the prediction and to send it to the data visualizer. It may or may not be on the same machine as the data server.

2.2.4. Display system

Display the results in the form of graphs for easy analysis.

Capítulo 3

Nodo

3.1. Introducción

Como hemos comentado anteriormente, el nodo es quien recoge los datos meteorológicos escogidos y los envía al servidor de datos.

Se ha implementado con una Raspberry Pi 3 como placa programable, los sensores DHT22 para la medición de la temperatura y humedad, y un piranómetro SP-212 para medir la irradiación solar.

La parte software del nodo ha sido implementada en Python.

3.2. Elementos hardware utilizados

3.2.1. Raspberry Pi 3

Raspberry Pi es un microcontrolador, un computador de placa reducida 3.1. Con él se pueden controlar distintos componentes electrónicos a través de los pines que expone y combinarlo con la potencia y versatilidad de un sistema operativo.

Raspberry proporciona el sistema operativo oficial llamado Raspbian, una distribución de Debian. También soporta otros sistemas operativos.

Fue creado en 2006, en la Universidad de Cambridge, con el fin de fomentar la enseñanza en las escuela de ciencias de la computación, pero hasta 2012 no salió al mercado.

Existen distintos modelos de Raspberry Pi, desde la Raspberry Pi 1 Modelo A hasta la

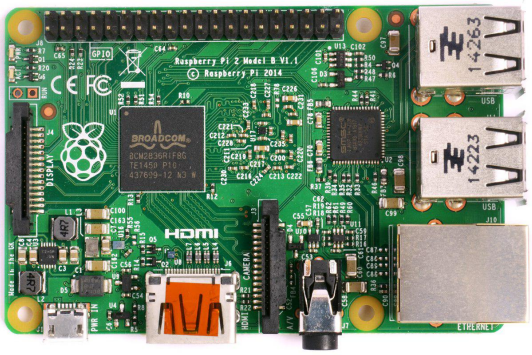

Figura 3.1: *Raspberry Pi 3 [Fuente: Raspberry Pi]*

Raspberry Pi 3 Modelo B 3.2. En nuestro proyecto hemos usado una Raspberry Pi 3 modelo B, con su sistema operativo oficial, Raspbian.

Se ha decidido utilizar este microcontrolador en el proyecto debido a su facilidad de manejo gracias a que cuenta con un sistema operativo y por ser relativamente barato para realizar un prototipo. Además cuenta con un gran comunidad detrás que trabaja en el desarrollo de nuevas guías y librerías para facilitar el desarrollo. ([7])

3.2.2. Sensor de temperatura y humedad

A la Raspberry se le conectan dos sensores, uno de temperatura y humedad y un piranómetro del cual hablaremos más tarde. Ambos para recoger los datos meteorológicos que necesitamos para nuestro modelo.

Para medir la temperatura y la humedad hemos utilizado un solo componente que recoge ambas muestras. El modelo escogido ha sido el sensor DHT22. ([1])

Algunas de las características de este sensor digital, y en especial de este modelo (ya que tambien existe el modelo DHT11 de la misma marca), son que tiene una precisión de 0.5ºC para medir la temperatura, y entre un 2 y un 5 por ciento para la humedad. Además recoge dos muestras por segundo. Ver figura 3.4.

La comunicación entre el sensor y la Raspberry se realiza a través de un pin GPIO (General Purpose Input/Output, Entrada/Salida) que es un pin de propósito general. Estos

	Raspberry Pi 1 Modelo A	Raspberry Pi 1 Modelo B	Raspberry Pi 1 Modelo B+	Raspberry Pi 2 Modelo B	Raspberry Pi 3 Modelo B
SoC:[5]	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)[3]			Broadcom BCM2836 (CPU + GPU + DSP + SDRAM + Puerto USB)	Broadcom BCM2837 (CPU + GPU + DSP + SDRAM + Puerto USB
CPU:	ARM 1176JZF-S a 700 MHz (familia ARM11)[3]			900 MHz quad-core ARM Cortex A7	1.2GHz 64-bit quad-core ARMv8
Juego de instrucciones:	RISC de 32 bits				
GPU:	Broadcom VideoCore IV,,[61] OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia),[59] 1080p30 H.264/MPEG-4 AVC[3]				
Memoria (SDRAM):	256 MiB (compartidos con la GPU)	512 MiB (compartidos con la GPU)[4] desde el 15 de octubre de 2012		1 GB (compartidos con la GPU)	
Puertos USB 2.0:[55]	1	2 (vía hub USB integrado)[54]	4		
Entradas de vídeo:[62]	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF				
Salidas de vídeo:[5]	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4),[63] Interfaz DSI para panel LCD[64] [65]				
Salidas de audio:[5]	Conector de 3.5 mm, HDMI				
Almacenamiento integrado:	SD / MMC / ranura para SDIO		MicroSD		
Conectividad de red:[5]	Ninguna	10/100 Ethernet (RJ-45) via hub USB[54]			10/100 Ethernet (RJ-45) vía hub USB,[66] Wifi 802.11n, Bluetooth 4.1
Periféricos de bajo nivel:	8 x GPIO, SPI, I²C, UART[61]			17 x GPIO y un bus HAT ID	
Reloj en tiempo real:[5]	Ninguno				
Consumo energético:	500 mA, (2.5 W)[5]	700 mA, (3.5 W)	600 mA, (3.0 W)	800 mA, (4.0 W)	
Fuente de alimentación:[5]	5 V vía Micro USB o GPIO header				
Dimensiones:	85.60mm × 53.98mm[67] (3.370 × 2.125 inch)				
Sistemas operativos soportados:	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux, SUSE [68] Linux Enterprise Server for ARM. RISC OS[2]				

Figura 3.2: *Cuadro comparativo de las especificaciones técnicas [Fuente: Wikipedia]*

pines pueden ser de entrada o de salida, y reciben y envían valores binarios de un bit.

En el datasheet del sensor viene definido el proceso de comunicación entre este y la Raspberry. Para ello, el microcontrolador (Raspberry en nuestro caso) envía una señal de arranque para que el sensor cambie del estado "standby" al estado "running". Al terminar de enviar dicha señal, el sensor envía una respuesta de 40 bits que contiene la información relativa a la humedad (16 bits), a la temperatura (16 bits) y un checksum (8 bits) para comprobar que se ha enviado correctamente. Ver imagen 3.3.

El envío de estos bits sucede de la siguiente forma: el sensor envía una señal "baja" ("0") que indica el inicio de la transmisión de un bit. Seguidamente envía una señal "alta" ("1"). Si la transmisión de esta corriente dura más de 50μ s indica que se ha transmitido un 1, en caso contrario, se quiso transmitir un 0.

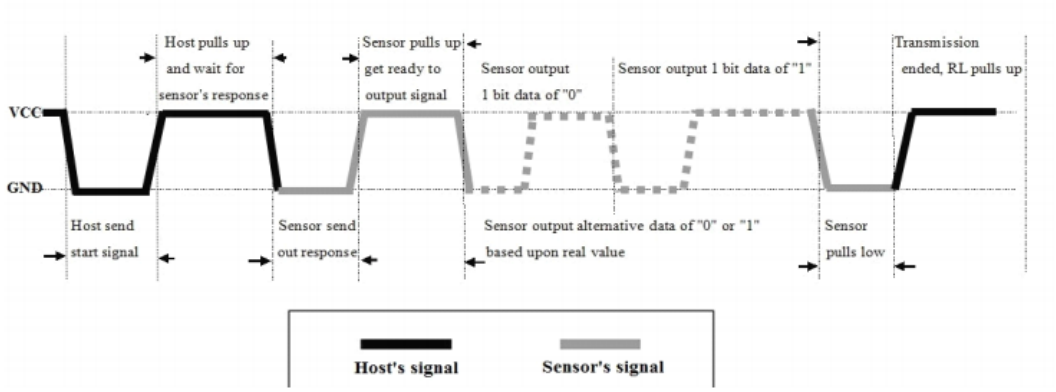

Figura 3.3: *Diagrama de comunicación del sensor DHT22 [Fuente: Adafruit]*

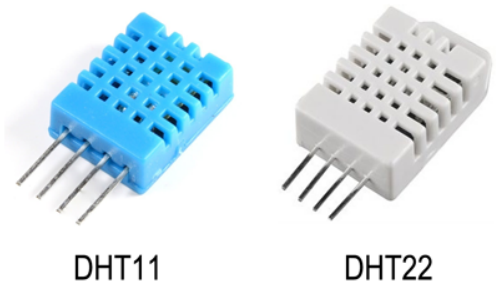

Figura 3.4: *Sensores de temperatura y humedad DHT11 y DHT22 [Fuente: Adafruit]*

3.2.3. Piranómetro y ADC

Para recoger la cantidad de irradiación solar se utiliza un piranómetro. En este proyecto se ha utilizado el modelo SP-212. Este sensor es analógico, es decir, la señal que envía es un voltaje variable con el cual indica la cantidad de irradiación que recoge. ([2]).

Para poder leer esa información con un microcontrolador es necesario hacer una conversión a digital. Para ello se utiliza un conversor analógico-digital (ADC).

El ADC utilizado en este proyecto ha sido el MCP3008. Es un conversor de 10 bits por lo que permite codificar entre 0 y 1024. Se comunica con la Raspberry a través del puerto SPI. ([1])

SPI es un estándar de comunicación en serie, maestro/esclavo, regulado por un reloj.

Incluye 4 señales: reloj (SCLK), salida de datos del maestro y entrada de datos al esclavo (MOSI), salida de datos del esclavo y entrada al maestro (MISO) y otra para seleccionar un esclavo, o para que el maestro le diga al esclavo que se active (SS/Select).

Con cada pulso de reloj el maestro envía un bit. Para que empiece la transmisión el maestro, baja la señal SS/Select a cero, con esto el esclavo se activa y empieza la transmisión, con un pulso de reloj al mismo tiempo que el primer bit es leído.

Raspberry cuenta con unos pines para la comunicación a través de SPI lo que facilita el uso de este tipo de componentes.

3.3. Componentes software y librerías externas

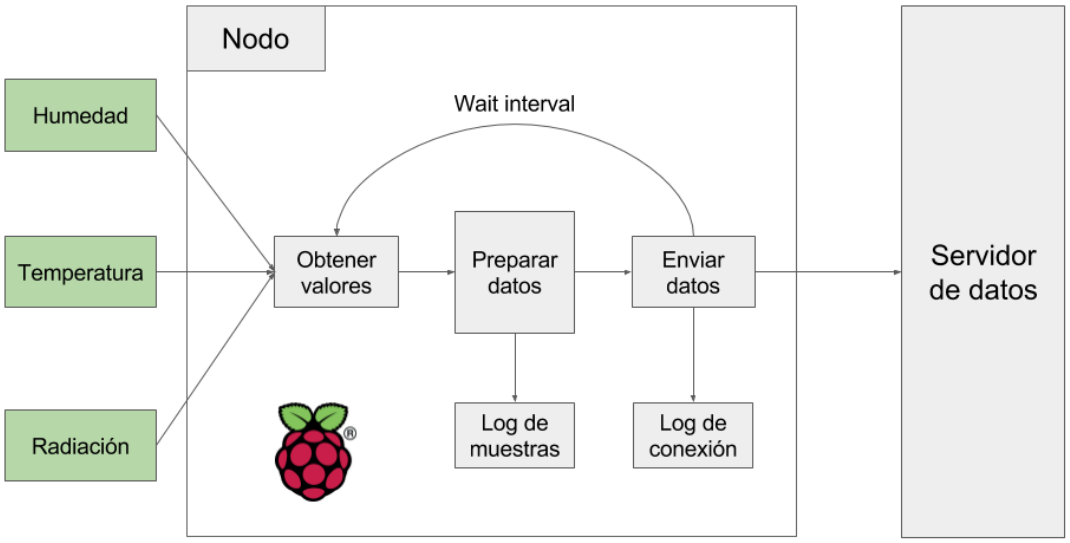

Figura 3.5: *SW nodo*

El software a través del cual controlar el nodo ha sido escrito en Python2.7. Su punto de entrada es el archivo "solar_node.py".

Se puede ejecutar a través de la consola de comandos con el comando "python solar_node.py"

Este programa tiene dos partes importantes: recoger los datos y enviarlos al servidor de datos. Para ello utilizamos varias librerías ya existentes.

Los datos recogidos serán almacenados en registros para evitar la pérdida de datos. Estos registros (o "logs") son almacenados en archivos diarios dentro del nodo. El resultado de las conexiones con el servidor de datos también serán registradas en un fichero guardando si la conexión se realizó con éxito o no.

Existen diferentes argumentos posibles para cambiar el modo de ejecución del nodo.

- start: arranca el nodo en modo servicio. Toma muestras de los sensores cada cierto periodo de tiempo (por defecto un minuto) y las almacena en los registros.
- test: arranca el nodo para realizar una sola muestra. El fin de este argumento es comprobar el correcto funcionamiento del nodo.
- help: muestra la ayuda del comando con todas las opciones disponibles.
- m: (opcional) los datos recogidos por el nodo serán enviados al broker MQTT.
- i [number]: (opcional) especifica el intervalo de recogida de muestras de los sensores.

3.3.1. Librerías externas

- Adafruit_DHT: librería proporcionada por Adafruit para leer e interpretar los datos proporcionados por el sensor DHT22. Adafruit es una compañía de hardware open-source, que además de proporcionar hardware, suministra de una gran cantidad de documentación y librerías para facilitar el trabajo con sus componentes.
- Adafruit_GPIO: aporta la funcionalidad para acceder a los pines GPIO a través de python.
- Adafruit_MCP3008: comunicación a través de SPI con el ADC

- paho-mqtt: Paho ha sido creado para proporcionar implementaciones escalables de código abierto de protocolos de mensajería abiertos y estándar dirigidos a aplicaciones nuevas, existentes y emergentes para Machine to Machine (M2M) e Internet of Things (IoT). Esta librería proporciona la funcionalidad para la comunicación a través de MQTT. Este protocolo de comunicación será exlicado más adelante.

3.3.2. Comunicación

La comunicación del nodo con el resto del sistema se realiza a través del protocolo MQTT (explicado en el punto 4.2) sobre la tecnología inalámbrica Wi-Fi. Debido a que el proyecto es un prototipo, y a que la Raspberry 3 incluye un módulo de Wi-Fi, no nos hemos preocupado por la optimización de recursos. De ser así, habríamos elegido otro tipo de tecnologías usadas en comunicaciones "machine-to-machine", que consumen menos recursos como por ejemplo LoRa.

LoRa (Long Range) es una tecnología que ofrece un largo alcance, un bajo consumo de energía y una transmisión segura de datos. Entre sus ventajas cabe señalar que permite aplicaciones de seguimiento sin GPS, tiene un bajo coste y una alta capacidad, ideal para operadores de redes públicas.

Capítulo 4

Servidor de datos

4.1. Introducción

La información recogida por el nodo es almacenada en un servidor central ("servidor de datos"). Este se encarga de recibir la información proporcionada por uno o varios clientes (el nodo en nuestro caso), almacenarla y distribuirla a quién la requiera.

La comunicación se realiza sobre el protocolo MQTT, muy utilizado para la comunicación máquina a máquina. Como implementación de este protocolo utilizamos el servidor mosquitto.

Existen varias alternativas a esta tecnología como por ejemplo:

- CoAP (Constrained Application Protocol) es un protocolo que está diseñado para la transferencia de documentos entre dispositivos restringidos. Dentro de sus ventajas podemos destacar que está destinado para su uso en dispositivos de Internet con recursos limitados, está diseñado para traducir fácilmente a HTTP.

- AMQP (Advanced Message Queuing Protocol) destaca por su orientación a la mensajería de negocios. Regula el comportamiento tanto del servidor como del cliente de la mensajería logrando que las implementaciones de diferentes proveedores sean interoperables. Esta es una de sus características más señaladas.

Al final nos decantamos por MQTT debido a sus ventajas frente a las alternativas expuestas:

- Sistema de suscripción. Esto permite poder tener muchos nodos sin un gran esfuerzo.
- Utiliza un ancho de banda mínimo. Tiene una cabecera fija de 2 bytes más una cabecera variable. De este variable, durante la suscripción y conexión se utilizan 12 bytes. La mayoría de publicaciones utilizan 2 bytes de cabeceras variables.
- Consume muy poca energía.
- Permite una gran fiabilidad si esta es necesaria, por la definición de la QoS (Calidad de Servicio).
- Es muy escalable.

4.2. MQTT

MQTT o lo que es lo mismo *Message Queue Telemetry Transport*, es un protocolo para la comunicación "machine-to-machine" de mensajería muy simple y ligero, orientado a conexión por TCP/IP. Está diseñado principalmente para dispositivos con poco ancho de banda y con latencia baja.

MQTT es un servicio de publicación/suscripción TCP/IP sencillo y muy ligero. Se basa en el principio cliente/servidor y suscriptor/publicador.

El servidor, llamado broker, recopila los datos que los publicadores ("publishers" en adelante) le transmiten. Determinados datos recopilados por el broker se enviarán a determinados publishers que previamente así se lo hayan solicitado al broker.

Los publishers envían los mensajes a un canal llamado topic. Los suscriptores ("subscribers") pueden leer esos mensajes. Los topics (o canales de información) pueden estar distribuidos jerárquicamente de forma que se puedan seleccionar exactamente las informaciones que se desean.

La disposición jerárquica de estos "topics" está especialmente diseñada para facilitar la distribución de publishers y subscribers por grupos. De esta forma, si tuvieramos una instalación domótica, podríamos distribuir los elementos a controlar en grupos jerárquicos.

Los mensajes enviados por los dispositivos comunicantes pueden ser de todo tipo pero no pueden superar los 256 Mb.

4.2.1. Seguridad

Los datos de IoT intercambiados pueden resultar muy críticos, por lo que es posible garantizar la seguridad de los intercambios en varios niveles:

- Transporte en SSL/TLS.
- Autenticación mediante certificados SSL/TLS.
- Autenticación mediante usuario y contraseña.

4.2.2. QoS

El protocolo MQTT define tres modos diferentes de asegurar la Calidad del Servicio (QoS, Quality of Service) que el publicador podrá definir según sus necesidades.

Hay tres niveles posibles:

- QoS nivel 0 (At Most Once) se entregará como mucho una vez, por lo que el mensaje no tiene garantías de recepción (el broker no informa al publicador de que ha recibido el mensaje).
- Qos nivel 1 (At Least Once) se entregará al menos una vez. El publicador lo transmitirá varias veces si es necesario, hasta que el broker le confirme que lo ha enviado a la red. Este es el nivel por defecto.
- QoS nivel 2 (Exactly Once) será obligatoriamente guardado por el emisor, que lo transmitirá siempre que el receptor no confirme su envío a la red.

  En este proyecto se usa QoS nivel 1, que es el que tiene "mosquitto" por defecto.

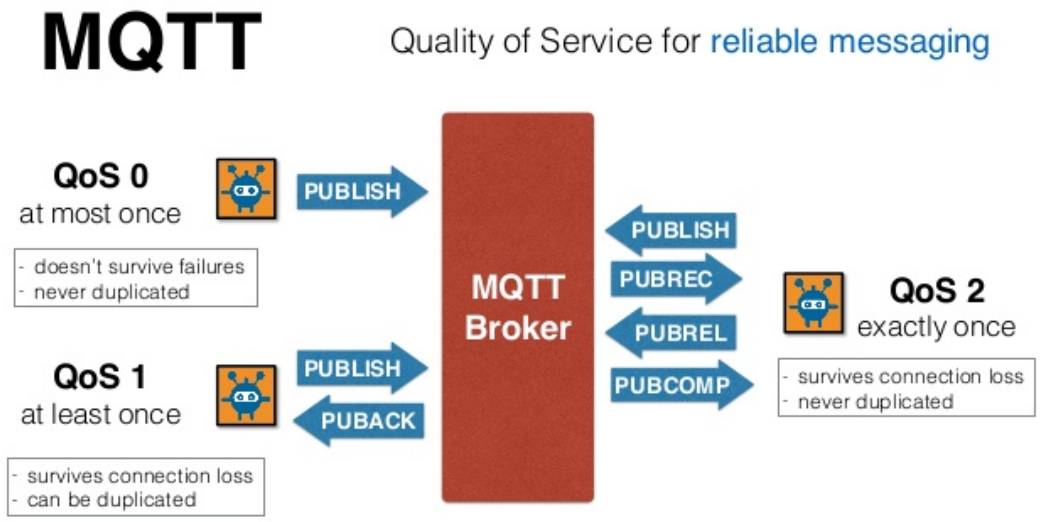

Figura 4.1: *Diagrama de los distintos niveles de calidad [Fuente: SlideShare: MQTT - A practical protocol for the Internet of Thing]*

Capítulo 5

Exploración del modelo predictivo

5.1. Introducción a los modelos predictivos

Cuando hablamos de modelos predictivos nos estamos refiriendo al análisis que agrupa una serie de técnicas para conseguir analizar datos reales tanto actuales como históricos y, así, conseguir predicciones. Con esta predicción podremos saber cuánto se acerca nuestro modelo a la realidad.

Existen algunos conceptos clave a la hora de hablar de metodos predictivos:

- Conjunto de entrenamiento: son los datos a partir de los cuales se desarrollan los modelos predictivos.
- Conjunto de validación: datos utilizados para evaluar el rendimiento de dichos modelos.
- Predictores: variables de entrada, utilizadas para la ecuación de predicción.
- Resultado: objetivo que se quiere predecir.

Las fases para desarrollar este tipo de modelos son las siguientes:

- Entrenamiento: para predecir con estas técnicas primero se debe "entrenar" el sitema. Esto se puede realizar de dos formas, proporcionando al entrenamiento los resultados ya conocidos de las predicciones (aprendizaje supervisado) o no (aprendizaje no supervisado). Más adelante se explicará en qué consiste este entrenamiento.

En este proyecto se utilizan algoritmos de aprendizaje supervisado.

Para realizar un "entrenamiento supervisado" se selecciona un conjunto de datos (conjunto de entrenamiento) del que se conoce el valor real de la predicción que queremos realizar y se explora, en función del algoritmo seleccionado, variando los parámetros internos del algoritmo para obtener un posible modelo.

Cada modelo tiene una serie de variables que podemos definir previas al entrenamiento que influirán en el resultado. En esta fase, los factores que se buscan explorar son aquellos que, internamente, ajusta el propio algoritmo en función de los datos de entrada y los parámetros que definamos.

- Validación: se selecciona otro conjunto de datos distinto al anterior (conjunto de validación) del que también se conoce el valor real de la predicción y se aplica el modelo obtenido en el entrenamiento. Se comparan los datos proporcionados con el modelo con los datos reales y se sacan distintas métricas para conocer su efectividad.

  En función del resultado obtenido en la validación, se pueden volver a repetir estos dos pasos cambiando el conjunto de datos o los parámetros de entrada del modelo hasta conseguir una predicción óptima.

- Predicción: una vez encontrado un modelo en la fase anterior que cumpla con los requisitos, se guarda ese modelo en algún formato recuperable para poder utilizarlo en cualquier momento sin tener que volver a entrenarlo.

Los algoritmos predictivos que se han estudiado han sido los siguientes:

- Regresión Lineal
- SVR (Support Vector Regression)
- Redes Neuronales

5.2. Conjunto de datos

Para poder encontrar el modelo más adecuado para este problema se necesita una gran cantidad de datos con la que entrenar y validar. Estos datos han sido obtenidos de InfoRiego, una herramienta pensada para estudiar el consumo de agua en los cultivos y hacerlo más eficiente que proporciona la información meteorológica que necesaria para este proyecto.

"InfoRiego" hace uso de la red de estaciones meteorológicas SIAR (proyecto de la Dirección General de Desarrollo Rural del Ministerio de Medio Ambiente y Medio Rural y Marino) y de las estaciones del ITACyL (Instituto Tecnológico Agrario de Castilla y León).

Hay dos formas de obtener los datos de InfoRiego: a través de una API Rest o a través de FTP.

Este proyecto realiza la consulta de datos a través del servidor FTP de inforiego. Este servidor proporciona acceso mediante HTTP por lo que se obtienen los archivos con la información meteorológica deseada a través de WebScrapping.

WebScrapping es una técnica que recorre, a través de software, los elementos HTML de un sitio web pudiendo así obtener su información.

Se ha creado un script en Python que, dados unos parámetros de consulta (fecha y hora de inicio, fecha y hora de fin y ubicación) obtiene, a través de WebScrapping, los archivos CSV almacenados en InfoRiego que cumplen con los parámetros y los combina en un solo archivo CSV con toda la información. Ver figura 5.1

Este archivo CSV resultante (o el que se requiera en cada momento), es el utilizado como conjunto de entrenamiento y validación. La automatización de esta tarea permite que la exploración del modelo se pueda realizar de forma sencilla, pudiendo realizar así muchas más pruebas.

Los datos proporcionados a cualquier modelo predictivo son previamente normalizados. Es importante que todos estén en la misma escala ya que, de no ser así, se puede interpretar que los datos en una mayor escala tienen un mayor peso dentro de la predicción.

Para dicha normalización, son necesarios conocer el máximo, el mínimo y la media del

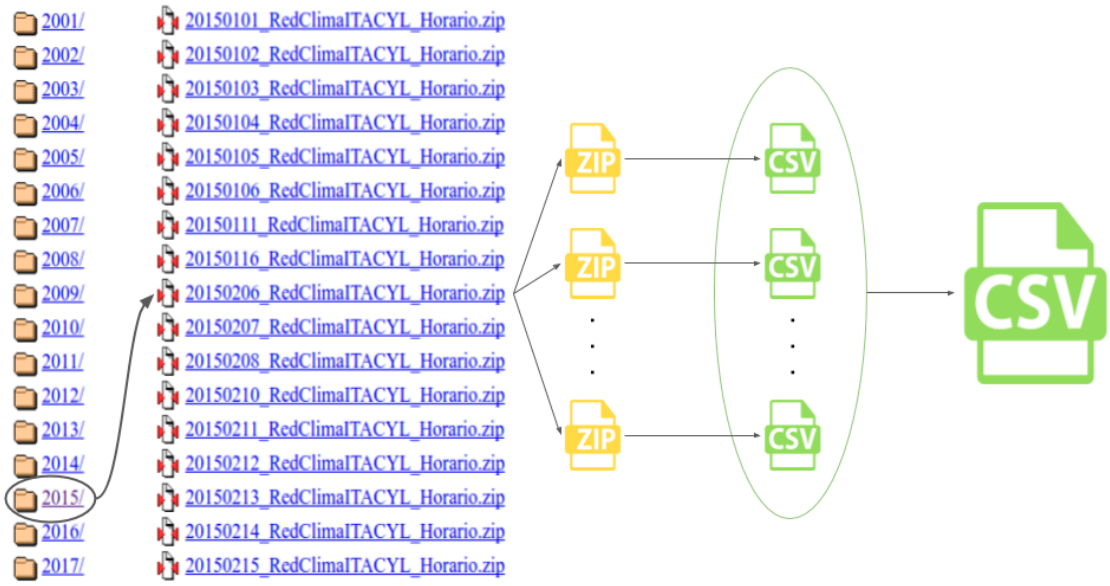

Figura 5.1: *Flujo de obtención de datos de InfoRiego*

conjunto de datos a normalizar. Estos datos serán almacenados para, posteriormente, poder invertir el proceso y obtener el valor inicial.

Este conjunto de datos debe sufrir una transformación más antes de poder ser utilizado en el entrenamiento. El entrenamiento y la validación necesitan dos elementos cada uno: los datos con los que se realiza la predicción y el resultado real que debe obtener. A estos dos elementos los llamaremos conjunto X y conjunto Y. Ver figura 5.3.

Ambos conjuntos parten del mismo conjunto de datos generado anteriormente.

El conjunto X tiene una serie de registros. Cada uno de ellos representa los datos necesarios para realizar una predicción. En este caso, son necesarias la fecha, hora, temperatura, humedad y radiación de varios instantes anteriores para realizar una predicción en un instante posterior.

El número de instantes anteriores lo llamaremos, de aquí en adelante, "k" por lo que serán necesarias las "k" muestras anteriores de hora, temperatura, humedad y radiación (la fecha y la ubicación serán la misma para todos los instantes anteriores). Indicando varios

Normalización:

$$X_{normalizado} = X - media / (máximo - mínimo)$$

Revertir normalización:

$$X = X_{normalizado} * (máximo - mínimo) + media$$

Figura 5.2: *Fórmulas de normalización*

instantes anteriores conseguimos que cada predicción se realize con una "tendencia" que se espera que el algoritmo puede ser capáz de inferir.

Al instante posterior al momento de realizar la predicción se llama "horizonte de predicción" y es el instante de tiempo para el que se quiere conocer la predicción.

Cada registro del conjunto Y indica la predicción que debe obtener cada registro en el mismo índice del conjunto X. Para obtener este valor, se recorre el conjunto X y se toma la radiación que ocurrió "n" instantes después en el conjunto de datos principal.

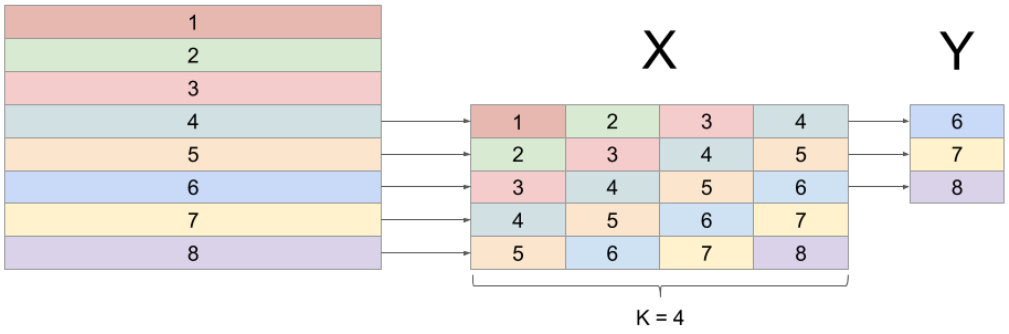

Figura 5.3: *Transformación de datos para entrenamiento*

5.3. Algoritmos utilizados

5.3.1. Regresión Lineal

La regresión lineal es un modelo matemático que trata de hallar los coeficientes de una combinación lineal que mejor se ajusten a un conjunto de puntos dispersos conocidos a través del método de los mínimos cuadrados. Esto es, trazar una línea en el espacio que pase lo más próximo posible a todos los puntos.

Al conocer dicha función lineal, podremos "predecir" con cierto grado de exactitud, lo que pasará.

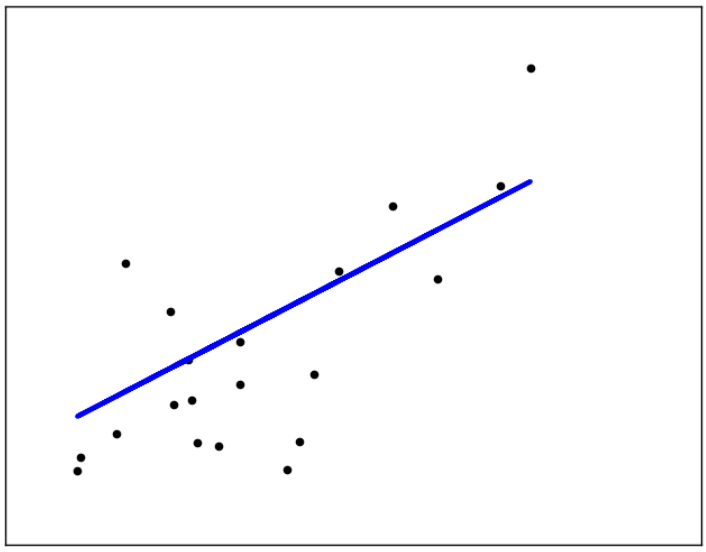

Figura 5.4: *Regresión lineal [Fuente: Wikipedia]*

La recta que se puede ver en la gráfica 5.4, muestra cómo la regresión lineal intenta dibujar una línea que minimice mejor la suma residual de cuadrados entre las respuestas observadas en el conjunto de datos, y las respuestas predichas por la aproximación lineal.

En este proyecto, la regresión lineal explorará nuestros datos con las predicciones ya conocidas buscando una regresión que más ajuste la predicción a los datos reales.

5.3.2. SVR (Support Vector Regression)

SRV es una nueva versión de SVM para regresión. SVM es un algoritmo de aprendizaje supervisado de la familia de los clasificadores.

El término clasificador se utiliza en referencia al algoritmo utilizado para asignar un elemento entrante no etiquetado en una categoría concreta conocida. Dicho algoritmo, permite pues, ordenar o disponer por clases elementos entrantes, a partir de cierta información característica de estos.

En SVM los datos son etiquetados en distintas clases y el clasificador intentará construir un hiperplano o conjunto de hiperplanos dentro del conjunto de datos que mejor separe las clases unas de otras. Ver figura 5.5.

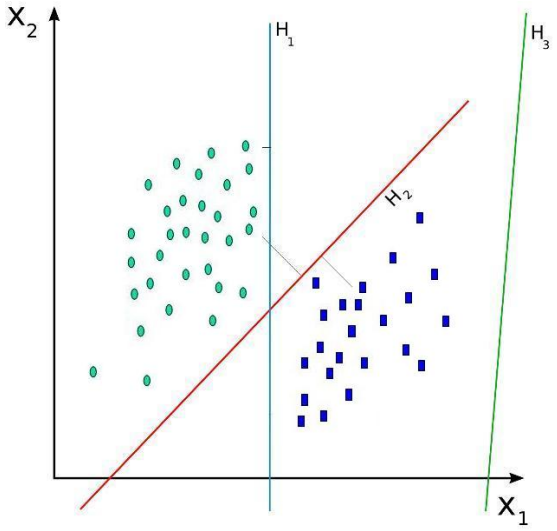

Figura 5.5: *SVM [Fuente: Wikipedia]*

Las características más notables de los clasificadores SVM son:

- Proyectar previamente los datos a un espacio de dimensionalidad superior.
- Buscar el hiperplano que tenga la máxima distancia con los puntos que estén más cerca de él mismo. Es decir, que dejen una mayor margen entre los datos.

La diferencia entre SVR y SVM es que SVR intenta hacer una regresión a partir del clasificador. Para esto, realiza un mapeo no lineal de los datos del entrenamiento a un espacio de mayor dimensión, donde podremos realizar una regresión lineal. Ver figura 5.6.

SVR permite varios modos de predicción "kernel": linear, poly, rbf, sigmoid... Este entrenamieto se ha llevado a cabo con los modos "linear" y "rbf" que, intuitivamente, pueden ser los que mas se acercan a nuestros datos reales.

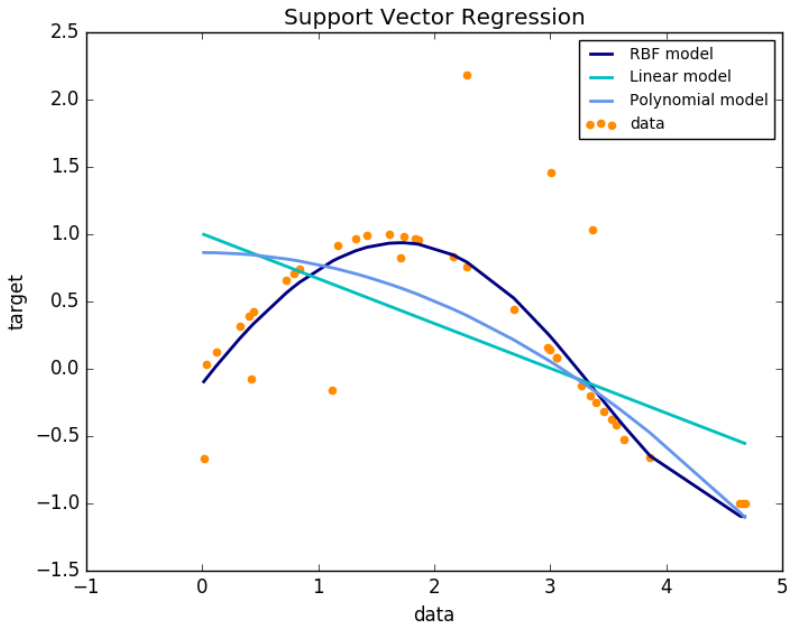

Figura 5.6: *SVR [Fuente: scikit learn]*

5.3.3. Redes Neuronales

Las redes neuronales son un algoritmo de aprendizaje supervisado de la familia de los clasificadores, al igual que SVR 5.3.2.

Se basan en el principio del funcionamiento de las neuronas del cuerpo humano. Cada neurona recibe varias señales y propaga, en función de estas, otro resultado al resto de la red.

Las neuronas de estas redes, son llamadas perceptrones. Son unidades funcionales sin una función específica que buscan aplicar un peso a las señales recibidas y en base a ello, envía un señal a otra neurona.

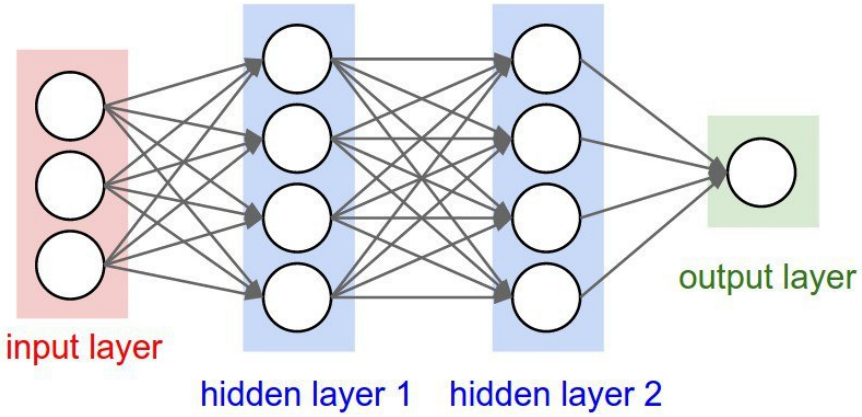

Figura 5.7: *Red neuronal [Fuente: scikit-learn]*

La distribución de las neuronas se realiza en distintas capas. La primera capa "capa de entrada" recibe los datos de entrada. La última capa proporciona los resultados de la clasificación, normalmente la clase a la que pertenecen los datos de entrada.

Las capas intermedias buscan aumentar la posibilidad de ajuste de los pesos. El ajuste de estos pesos se realiza de la siguiente forma:

- Cada neurona empieza con unos pesos aleatorios.
- Se introducen datos aleatorio dentro del conjunto total de entrada.
- La red genera un vector de datos de salida.
- Esta salida se compara con el resultado esperado.
- El error obtenido se propaga a la capa de neuronas anterior ("back-propagation") y se usa para ajustar los pesos.
- Se continua propagando el error hacia atrás ajustando los pesos de cada capa hasta alcanzar la capa de entrada.

- Esto se repetirá con diferentes datos de entrenamiento.

Es tarea del desarrollador la elección del número de capas y de neuronas en cada capa.

La diferencia más importante entre este algorítmo y los dos anteriores es que no es una regresión, si no un clasificador. Esto implica que cada radiación que se introduzca en el modelo debe estar etiquetada en una clase. Para etiquetar la radiación en distintas clases se ha seguido el siguiente procedimiento:

- Recordemos que, como está indicado en el capítulo anterior (capítulo 4), los datos del conjunto de datos han sido normalizados entre -1 y 1. Esto facilita la tarea ya que conocemos el máximo y el mínimo de nuestro conjunto.
- Se definen 100 clases, y se etiqueta la radiación en una de ellas.
- Para etiquetar, se utiliza la siguiente fórmula: "f(x) = (x + 1) / 2 * 100" y se conserva solo la parte entera del resultado.
- Esto transforma los datos de [-1, 1] a [0, 1] y, al multiplicarlo por 100, se obtiene la clase deseada.

5.4. Método de estudio

5.4.1. Grid Search

Para optimizar la elección del modelo predictivo y sus parámetros, necesitamos estudiar el comportamiento que tiene un modelo con todos sus posibles valores en cada parámetro, teniendo así la certeza de que elegimos la mejor configuración.

Esta es una tarea que no se puede llevar a cabo debido a su complejidad computacional, por lo que existen distintas maneras de abordar el problema. En respuesta a esto, surgen métodos de análisis más optimos, entre ellos encontramos Grid Search.

Grid Search es un método de estudio ya implementado al que se le especifica los algoritmos a estudiar y rangos para los distintos parámetros de cada uno.

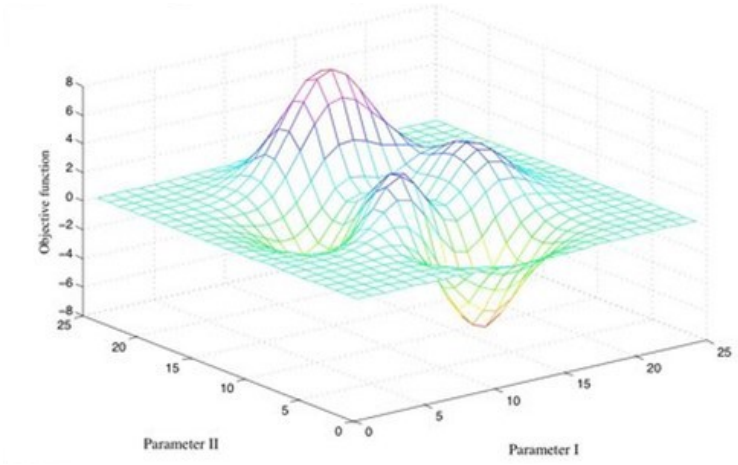

Figura 5.8: *Grid Search [Fuente: scikit-learn]*

Este método observa qué muestra del rango es la que obtiene un "mejor resultado" y vuelve a realizar el estudio "haciendo zoom" con centro en esta muestra.

Para calcular el mejor resultado se utiliza una técnica llamada "K-Fold Cross-Validation". Esta técnica divide los datos en "k" conjuntos (no confundir esta "k" con la k que se utiliza para especificar los instantes de tiempo anteriores de cada predicción). Uno de ellos es utilizado como datos de prueba y el resto como datos de entrenamiento. Se realiza un proceso de validación y se repite k veces con cada uno de los posibles subconjuntos. Ver figura 5.9.

Al realizar este proceso de validación con datos conocidos, puede conocer la calidad del modelo predictivo con distintas métricas. En este proyecto se utilizan las métricas "R^2" y "accuracy" que serán explicadas en el apartado de resultados.

Una vez realizado el estudio con los distintos modelos y parámetros se puede observar cual es la mejor configuración para nuestros datos.

Existen otros métodos de exploración del modelo predictivo como por ejemplo "Randomized Parameter Optimization", que realiza el estudio probando con configuraciones aleatorias.

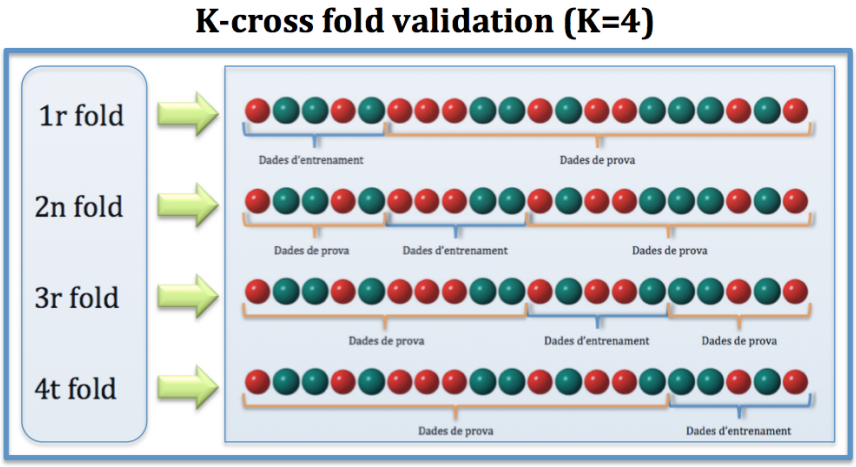

Figura 5.9: *Validación cruzada [Fuente: scikit-learn]*

5.4.2. Sistema de entrenamiento

Para llevar a cabo la implementación de Grid Search, hemos utilizado las funciones proporcionadas por scikit-learn ([8]). Esta librería ofrece una implementación de Grid Search a través de una función que nos permite abstraer el algoritmo de los modelos predictivos.

Hemos desarrollado distintas baterías de pruebas a través de ficheros en formato JSON (JavaScript Object Notation) que definen los distintos modelos y parametros que debe explorar Grid Search. Estas baterias de prueba son procesadas por un algoritmo encargado de instanciar los distintos modelos y suministrárselos a Grid Search. Ver figura 5.10.

Los resultados obtenidos de cada prueba son almacenados en archivos con formato CSV para su posterior análisis. Ver figura 5.11.

Las columnas de los resultados son:

- Modelo: Nombre del algoritmo usado.
- k: Número de muestras anteriores utilizadas para predecir.
- Distancia: Número de muestras posteriores sobre la que se quiere encontrar la predicción.

```
{
    "id": "regression",
    "start": 20150101,
    "end": 20151231,
    "k": [2, 3, 4, 5],
    "estimator": [{
            "module": "sklearn.linear_model",
            "class": "linear_model",
            "model": "LinearRegression",
            "parameters": [{
                "fit_intercept": [true, false],
                "copy_X": [true, false]
            }]
        },
        {
            "module": "sklearn.svm",
            "class": "svm",
            "model": "SVR",
            "parameters": [{
                    "C": [1, 10, 100, 1000],
                    "kernel": ["linear"]
                },
                {
                    "C": [1, 10, 100, 1000],
                    "gamma": [0.001, 0.0001],
                    "kernel": ["rbf"]
                }
            ]
        }
    ]
}
```


Figura 5.10: *Ejemplo JSON de prueba*

- Media: Valor entre 0 y 1 que indica la precisión de la predicción.
- Error: Valor entre 0 y 1 que indica la desviación típica de esta predicción.
- Parámetros de la prueba: Configuración de los posibles valores con los que se han obtenido la media y el error.

Una vez elegido el modelo predictivo y su configuración, podemos volver a entrenar el sistema guardando el resultado de este entrenamiento. De esta forma, en el momento de

Modelo	k	distancia	Media	Error	Parámetros de la prueba
LinearRegression	5	2	0.910009	0.003789	{u'copy_X': True, u'fit_intercept': True}
LinearRegression	5	2	0.910009	0.003789	{u'copy_X': True, u'fit_intercept': False}
LinearRegression	5	2	0.910009	0.003789	{u'copy_X': False, u'fit_intercept': True}
LinearRegression	5	2	0.910009	0.003789	{u'copy_X': False, u'fit_intercept': False}
LinearRegression	4	2	0.905659	0.000913	{u'copy_X': True, u'fit_intercept': True}
LinearRegression	4	2	0.905659	0.000913	{u'copy_X': True, u'fit_intercept': False}
LinearRegression	4	2	0.905659	0.000913	{u'copy_X': False, u'fit_intercept': True}
LinearRegression	4	2	0.905659	0.000913	{u'copy_X': False, u'fit_intercept': False}
SVR	5	2	0.905100	0.004216	{u'kernel': u'rbf', u'C': 1000, u'gamma': 0.001}
SVR	4	2	0.901516	0.000330	{u'kernel': u'rbf', u'C': 1000, u'gamma': 0.001}
SVR	5	2	0.901323	0.004353	{u'kernel': u'linear', u'C': 1}
SVR	5	2	0.901292	0.003694	{u'kernel': u'linear', u'C': 1000}

Figura 5.11: *Ejemplo de resultados Grid Search*

la predicción no necesitaremos volver a entrenar porque los parámetros del modelo están ajustados.

5.5. Descripción de las librerías usadas

5.5.1. Scikit-learn

Scikit-learn es una biblioteca de aprendizaje de software libre para el lenguaje de programación Python. Cuenta con varios algoritmos de clasificación, regresión y agrupación, incluyendo máquinas de vector de apoyo, bosques aleatorios y aumento de gradiente, y está diseñado para interoperar con las bibliotecas numéricas y científicas Python NumPy y SciPy. ([8])

5.5.2. NumPy

NumPy es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices. ([4])

Capítulo 6

Predicción y visualización

6.1. Servidor de predicción

Aunque este servidor sea un componente autónomo del sistema puede estar alojado en la misma máquina que el servidor de datos. Aún así, la comunicación con este se hará a través de un cliente MQTT 4.2 para facilitar la posibilidad de distribuir el sistema en otra topología.

El cliente MQTT estará escuchando continuamente al "broker", y en el momento en el que reciba datos, los guardará en un registro diario. Ver figura 6.1.

codigo	fecha	hora	temperatura	humedad	radiacion
1	20150101.0	100.0	-3.97	86.2	0.0
1	20150101.0	130.0	-3.82	85.4	0.0
1	20150101.0	200.0	-4.42	84.8	0.0
1	20150101.0	230.0	-4.91	85.2	0.0
1	20150101.0	300.0	-4.98	89.0	0.0
1	20150101.0	330.0	-4.51	87.4	0.0
1	20150101.0	400.0	-4.71	87.8	0.0
1	20150101.0	430.0	-5.69	88.9	0.0
1	20150101.0	500.0	-4.71	88.2	0.0
1	20150101.0	530.0	-6.27	88.4	0.0

Figura 6.1: *Log del servidor*

El resultado del entrenamiento del modelo predictivo escogido durante la fase de en-

trenamiento, ha sido exportado y guardado en este servidor. De esta forma, a la hora de predecir recuperamos este entrenamiento y evitamos volver a realizarlo.

Una vez obtenida la predicción esta es enviada, junto con el resto de datos que recibió el servidor, a nuestro sistema de visualización.

6.2. Sistema de visualización

Existen distintos sistemas disponibles en el mercado (IoTDataViz, FreeBoard, IBMBlumix...), o incluso podríamos desarrollar uno propio, pero nos hemos decantado por ThinkSpeak.

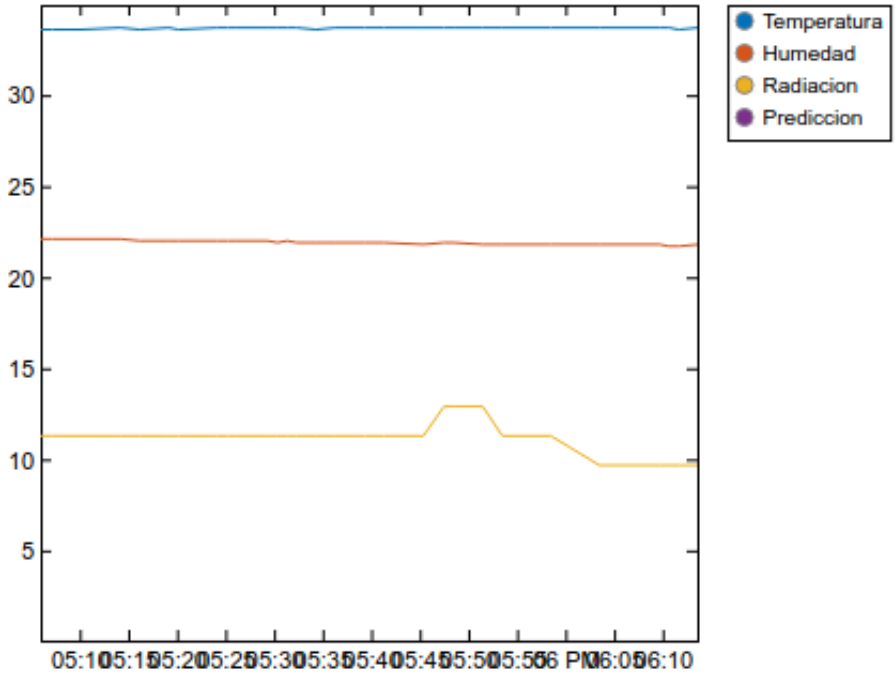

Figura 6.2: *Gráfica ThingSpeak*

ThinkSpeak es una plataforma online orientada al "Internet de las cosas" (IoT) pensada para almacenar y realizar un seguimiento de los datos obtenidos por distintas aplicaciones.

Una de sus principales ventajas es que los datos recibidos puedes ser procesados por

scripts propios implementados en MATLAB. Permitiendo así una observación y manipulación de los datos más personalizada. También cuenta con la ventaja de ser gratuita.

Actualmente, la comunicación con ThinkSpeak se puede realizar nativamente a través de MQTT, sin embargo en el momento de desarrollar la comunicación entre nuestro sistema y este, no contaba con esta característica.

La manera que proporcionaba ThinkSpeak para esta comunicación era a través de HTTP a través de una API REST. La ventaja que tiene esto, es que cualquier lenguaje que soporte manejo de peticiones HTTP, puede comunicarse con él, permitiéndonos así construir un sistema más modular.

Para realizar el envío de datos a través de esta API, basta con realizar una petición HTTP POST que lleve en el "body" (cuerpo de la petición) un campo "key" con la clave proporcionada por el servicio y los datos que queremos visualizar.

Capítulo 7

Resultados

Los resultados de este apartado se basan en la exploración tipo Grid descrita en el apartado anterior. Los datos utilizados en esta exploración han sido los proporcionados por InfoRiego entre el 1 de enero de 2015 hasta el 31 de diciembre del 2015.

Para llevar a cabo el entrenamiento anterior, y poder obtener una métrica con la que evaluarlo, es necesario realizar una división del conjunto de datos en dos para cada prueba. Un conjunto para entrenar el modelo y otro para comprobar la precisión de este. Es importante que estos dos conjuntos sean complementarios ya que el modelo entrenado no debe conocer ningún dato del conjunto de test o el entrenamiento se vería alterado por conocer de antemano los datos a predecir.

El tamaño de cada conjunto es también un hecho relevante ya que con un conjunto de datos muy pequeño para el entrenamiento, la predicción sería dificilmente alcanzable con éxito y con un conjunto muy grande, habría pocos datos para probar que el modelo es fiable. Una división típica es dividir el conjunto de datos en 70/30, es decir, 70 % de los datos para entrenamiento y 30 % para comprobación. En este se ha utilizado una división 25/75 que es la utilizada por defecto por el grid search de SciKitLean.

Para poder analizar los resultados, la prueba anterior proporciona distintas métricas para medir la calidad del modelo. La implementación de GridSearch utilizada (librería de scikit-learn) pretende dar una misma interfaz para utilizar esta técnica con distintos algoritmos, por lo tanto, los resultados que se obtienen con ellos también tienen una misma forma de

nombrar las métricas para todos.

Sin embargo, aunque en los ficheros de resultados aparezcan los mismos nombres para medir los resultados, el significado de estos es distinto dependiendo del modelo seleccionado. En los distintos apartados dedicados a cada uno de estos modelos se explicará el significado de sus métricas.

Junto con las métricas de calidad del modelo entrenado aparecen los parámetros necesarios para obtener dicho modelo. Tanto los parámetros específicos de cada modelo como el número de instantes anteriores utilizados ("k") o la distancia a la predicción.

A la distancia a la predicción la llamaremos "horizonte de predicción" y especifica la diferencia entre la hora a la que se realiza la predicción y la hora a la que se quiere predecir entre el intervalo de medida (30 minutos para estos datos). Aunque se haya probado con varios horizontes, el que más nos interesa es "2", que indica una predicción a una hora.

Se han realizado tres conjuntos de pruebas, una por cada algoritmo predictivo estudiado. A continuación se analizan los resultados para cada uno de estos algoritmos.

7.1. Regresión Lineal

Los parámetros suministrados a GridSearch para realizar el entrenamiento con este modelo son los siguientes:

- Inicio: 01-01-2015
- Fin: 31-12-2015
- k: 2, 3, 4, 5
- Horizonte de predicción: 2, 3, 4, 5, 6, 7
- fit_intercept: true, false
- copy_X: true, false

Grid serach estudiará los resultados para todas las combinaciones posibles.

De los resultados obtenidos para este algoritmo mostrados en la figura 7.1, la columna "mean" nos indican el R^2 obtenido. R^2 es el coeficiente de correlación y, en regresión lineal, es el cuadrado del *coeficiente de correlación de Pearson*. Se calcula dividiendo el cuadrado de la covarianza del conjunto de datos reales y los predichos entre el producto de los cuadrados de la desviación típica de cada conjunto. Expresa la calidad de la predicción del modelo entre 0 y 1.

Nombre del modelo	K	Horizonte	Mean	Std	Parametros
LinearRegression	5	2	0.910009	0.003789	{u'copy_X': True, u'fit_intercept': True}
LinearRegression	5	2	0.910009	0.003789	{u'copy_X': True, u'fit_intercept': False}
LinearRegression	5	2	0.910009	0.003789	{u'copy_X': False, u'fit_intercept': True}
LinearRegression	5	2	0.910009	0.003789	{u'copy_X': False, u'fit_intercept': False}
LinearRegression	4	2	0.905659	0.000913	{u'copy_X': True, u'fit_intercept': True}
LinearRegression	4	2	0.905659	0.000913	{u'copy_X': True, u'fit_intercept': False}
LinearRegression	4	2	0.905659	0.000913	{u'copy_X': False, u'fit_intercept': True}
LinearRegression	4	2	0.905659	0.000913	{u'copy_X': False, u'fit_intercept': False}
LinearRegression	3	2	0.898350	0.008953	{u'copy_X': True, u'fit_intercept': True}
LinearRegression	3	2	0.898350	0.008953	{u'copy_X': True, u'fit_intercept': False}

Figura 7.1: *Resultado Regresión Lineal*

Los datos se muestran ordenados por la columna "mean" siendo así el primero de estos registros el que más coeficiente de correlación obtiene con un coeficiente de correlación de 0.910009.

Como se puede observar en la figura 7.1, el modelo obtenido con mejor coeficiente de correlación tiene la siguiente configuración:

- Inicio: 01-01-2015
- Fin: 31-12-2015
- k: 5
- Horizonte: 2
- copy_X: true

- fit_intercept: True

En el capítulo 12 (Anexo: Gráficas Regresión Lineal), se muestran distintas gráficas con la radiación de este modelo durante varios días del año, la predicción que obtenemos con el modelo obtenido con mayor coeficiente de correlación y el modelo conservador.

Este "modelo conservador" expresa una predicción de la radiación que se realiza asumiendo que en el instante *t+1* la radiación será la misma que la observada en el instante *t*. El objetivo de este proyecto es superar, como mínimo este modelo conservador.

Junto a la leyenda de nuestro modelo y del modelo conservador, encontramos las siglas "MAE" (Mean Absolute Error), que es la media de las diferencias entre una radiación real y su predicción en valor absoluto. Esto nos permite conocer la media del error expresado en la misma unidad que la radiación (W / m^2).

Como vistazo general, podemos observar que en todas las gráficas de resultados de regresión lineal, nuestro modelo obtiene un MAE mucho menor que el modelo conservador. Esto implica que nuestro modelo Regresión Lineal se ajusta mejor a la realidad.

En los días de primavera obtenemos los mejores resultados, en especial en las horas centrales del día. Quizá esto se deba a que la temperatura a estas horas es más estable que en el amanecer o atardecer.

Los días de verano e invierno, es dónde la diferencia de MAEs con respecto al modelo conservador es algo menor. Por lo que nuestra predicción sería levemente más acertada que dicho modelo.

7.2. SVR

Para la exploración del modelo SVR usamos los siguientes parámetros:

- Inicio: 01-01-2015
- Fin: 31-12-2015
- k: 2, 3, 4, 5

- Horizonte de predicción: 2, 3, 4, 5, 6, 7
- C: 1, 10, 100, 1000
- gamma: 0.001, 0.0001
- kernel: linear, rbf

En los resultados de la exploración mostrados en la figura 7.2, la columna "mean" utiliza la misma métrica que la vista en el apartado anterior. Esto es porque ambas son regresiones y esta es muy utilizada en ese caso.

Nombre del modelo	K	Horizonte	Mean	Std	Parámetros
SVR	5	2	0.905100	0.004216	{u'kernel': u'rbf', u'C': 1000, u'gamma': 0.001}
SVR	4	2	0.901516	0.000330	{u'kernel': u'rbf', u'C': 1000, u'gamma': 0.001}
SVR	5	2	0.901323	0.004353	{u'kernel': u'linear', u'C': 1}
SVR	5	2	0.901292	0.003694	{u'kernel': u'linear', u'C': 1000}
SVR	5	2	0.901253	0.003569	{u'kernel': u'linear', u'C': 100}
SVR	5	2	0.901101	0.003352	{u'kernel': u'linear', u'C': 10}
SVR	5	2	0.901038	0.004624	{u'kernel': u'rbf', u'C': 1000, u'gamma': 0.0001}
SVR	5	2	0.900175	0.004844	{u'kernel': u'rbf', u'C': 100, u'gamma': 0.001}
SVR	5	2	0.895941	0.004993	{u'kernel': u'rbf', u'C': 10, u'gamma': 0.001}
SVR	5	2	0.895813	0.005088	{u'kernel': u'rbf', u'C': 100, u'gamma': 0.0001}

Figura 7.2: *Resultado Regresión Lineal*

La figura 7.2 muestra el modelo obtenido con mejor coeficiente de correlación (0.905100) y su configuración:

- Inicio: 01-01-2015
- Fin: 31-12-2015
- k: 5
- Horizonte de predicción: 2
- C: 1000
- gamma: 0.0001

- kernel: rbf

En el capítulo 15 (Anexo: Gráficas SVR) podemos observar varias gráficas para distintos días del año con la predicción obtenida con este modelo frente a la radiación real y al modelo conservador.

En los días de invierno, obtenemos un mejor MAE que el modelo conservador. Podemos observar que en las horas centrales del día es cuando mejor se ajusta, pero no tanto como primavera u otoño. Cuando peor se ajusta es en las primeras horas del día, en el amanecer.

En los días de las estaciones de primavera y otoño, nuestro modelo SVR obtiene una mayor diferencia respecto al MAE conservador, siendo mucho menor. Así podemos concluir, que en estas estaciones, quizá, es cuando mejor precisión obtenemos con nuestro modelo.

En los días de verano, podemos observar que nuestro modelo es cuando menor precisión tiene con respecto a la realidad. Siendo así, la única estación del año en la que el modelo conservador tiene un mejor MAE.

7.3. Redes Neuronales

A continuación se indica los distintos parámetros con los que se ha explorado el modelo:

- Inicio: 01-01-2015
- Fin: 31-12-2015
- k: 2, 3, 4, 5
- Horizonte de predicción: 2, 3, 4, 5, 6, 7
- solver: lbfgs, sgd
- hidden_layers: (5, 2), (5, 3), (5, 4)
- ramdon_state: 1, 2

En este caso, la métrica utilizada para determinar la calidad del modelo ha sido "accuracy" (en español, exactitud). Es una métrica muy utilizada en clasificadores que indica el porcentaje de predicciones clasificadas correctamente.

Modelo	k	H	Media	Error	Parámetros de la prueba
MLPClassifier	3	2	0.571703	0.015103	{u'solver': u'lbfgs', u'random_state': 2, u'hidden_layer_sizes': [5, 4]}
MLPClassifier	4	2	0.566903	0.006241	{u'solver': u'lbfgs', u'random_state': 2, u'hidden_layer_sizes': [5, 4]}
MLPClassifier	3	2	0.560672	0.014781	{u'solver': u'lbfgs', u'random_state': 1, u'hidden_layer_sizes': [5, 2]}
MLPClassifier	4	2	0.56005	0.013945	{u'solver': u'lbfgs', u'random_state': 1, u'hidden_layer_sizes': [5, 3]}
MLPClassifier	3	2	0.558666	0.034928	{u'solver': u'lbfgs', u'random_state': 1, u'hidden_layer_sizes': [5, 3]}
MLPClassifier	4	3	0.555453	0.008553	{u'solver': u'lbfgs', u'random_state': 2, u'hidden_layer_sizes': [5, 3]}
MLPClassifier	4	2	0.555036	0.03963	{u'solver': u'lbfgs', u'random_state': 1, u'hidden_layer_sizes': [5, 4]}
MLPClassifier	3	4	0.551609	0.008575	{u'solver': u'lbfgs', u'random_state': 1, u'hidden_layer_sizes': [5, 4]}
MLPClassifier	4	3	0.551525	0.007992	{u'solver': u'lbfgs', u'random_state': 2, u'hidden_layer_sizes': [5, 4]}
MLPClassifier	3	4	0.551442	0.014907	{u'solver': u'lbfgs', u'random_state': 2, u'hidden_layer_sizes': [5, 4]}
MLPClassifier	2	3	0.551396	0.014749	{u'solver': u'lbfgs', u'random_state': 1, u'hidden_layer_sizes': [5, 3]}
MLPClassifier	4	2	0.551275	0.012431	{u'solver': u'lbfgs', u'random_state': 2, u'hidden_layer_sizes': [5, 2]}
MLPClassifier	4	2	0.550439	0.008413	{u'solver': u'lbfgs', u'random_state': 2, u'hidden_layer_sizes': [5, 3]}
MLPClassifier	3	2	0.550142	0.015901	{u'solver': u'lbfgs', u'random_state': 2, u'hidden_layer_sizes': [5, 3]}

Figura 7.3: *Resultado redes neuronales*

En la figura 7.3 podemos ver los resultados del grid search para Redes Neuronales. La configuración del modelo con mayor "accuracy" obtenido es la siguiente:

- Inicio: 01-01-2015
- Fin: 31-12-2015
- k: 3
- Horizonte de predicción: 2
- solver: lbfgs
- hidden_layers: (5, 4)
- ramdon_state: 2

En el capítulo 14 tenemos distintas gráficas con la predicción realizada por este modelo, la radiación real y la predicción del modelo conservador para distintos días del año 2016.

Las conclusiones finales con nuestro modelo de redes neuronales es que hay que seguir investigando dicho modelo y sus parámetros. Obtenemos un "accuracy" o acierto del 0 %. Por lo que no hay mucho más que decir.

Capítulo 8

Instalación y puesta en marcha

Para poner en funcionamiento el proyecto deberemos tener todos lo elementos que lo componen:

- Nodo.
- Servidor de datos.
- Servidor de cálculo. Puede estar alojado en la misma máquina que el servidor de datos.
- Sistema de visualización. En este caso, estar registrado en ThingSpeak y tener una clave de aplicación.

8.1. Nodo

8.1.1. Requisitos

- Raspberry Pi 3 model B con una distribución Linux (Raspbian).
- Conexión a Internet.
- Sensor DHT22.
- Conversor analógico-digital MCP3008.
- Piranómetro SP-212.

- SPI activo en la Raspberry con “raspi-config”.

El cableado de los componentes HW está descrito en la figura 8.1.

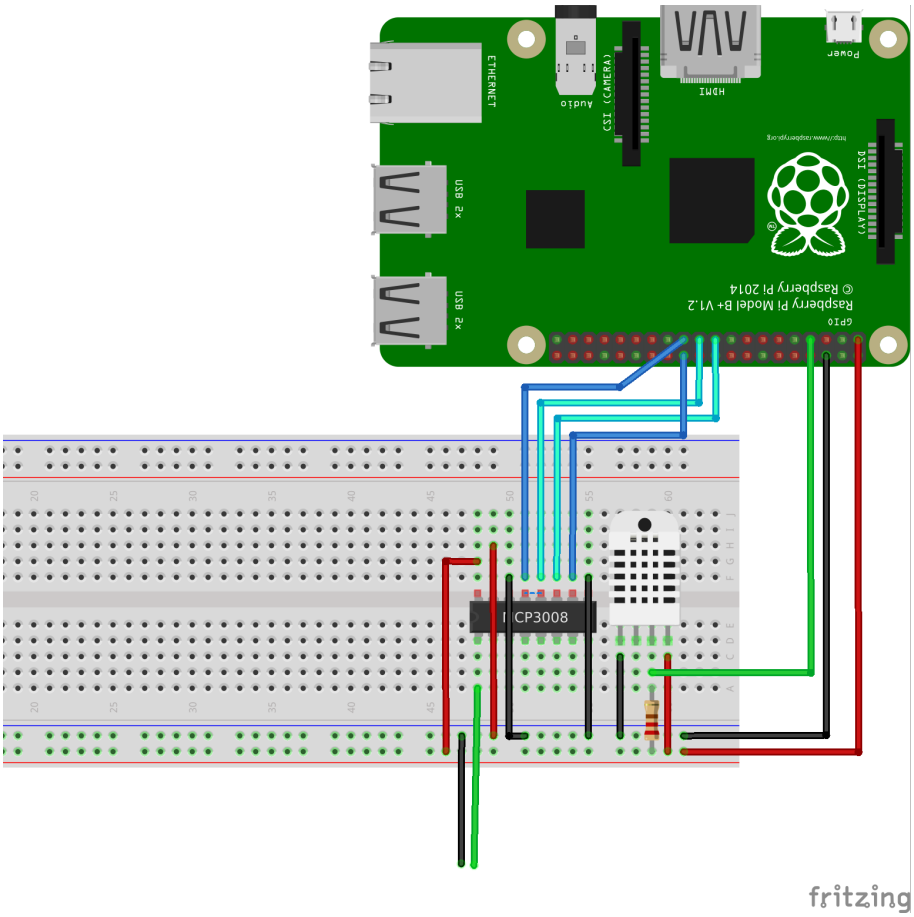

Figura 8.1: *Cableado del nodo*

8.1.2. Ficheros

Los archivos necesarios para la instalación del software del nodo se encuentran disponibles en Github.

- Introduce el directorio “node” en tu Rapsberry Pi

- Cambia las variables “brokerIp”, “brokerPort”, “topic”, “ubication”’ en solar_node.py

8.1.3. Dependencias

- Actualizar

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

- Python 2.7 ([6])

```
$ sudo apt-get install python2.7 build-essential
python-pip python-dev
```

- Paho ([5])

```
$ pip install paho-mqtt
```

- WiringPi ([9])

```
$ pip install wiringpi2
```

- Adafruit_DHT ([1])

```
$ git clone
https://github.com/adafruit/Adafruit_Python_DHT.git
$ cd Adafruit_Python_DHT
$ sudo python setup.py install
```

- Adafruit_GPIO

```
$ git clone
https://github.com/adafruit/Adafruit_Python_GPIO.git
$ cd Adafruit_Python_GPIO
$ sudo python setup.py install
```

- Adafruit_MCP3008

```
$ git clone
https://github.com/adafruit/Adafruit_Python_MCP3008.git
$ cd Adafruit_Python_MCP3008
$ sudo python setup.py install
```

8.1.4. Puesta en marcha del nodo

Para arrancar el nodo se deberá haber seguido todos los pasos anteriores. Posteriormente, desde un terminal de la Raspberry, ir a la carpeta donde se han guardado el código fuente y ejecutar el comando:

```
$ sudo python solar_node.py
```

Se debe ser superusuario para poder ejecutar este comando.

Para obtener más información sobre cómo utilizar el nodo ver Descripción del nodo o ejecutar:

```
$ sudo python solar_node.py -h
```

8.2. Servidor de datos

8.2.1. Requisitos previos

- Máquina con distribucion Linux instalada. Recomendable Debian, Fedora, OpenSUSE o Ubuntu.
- Conexión a internet.
- IP fija y opcionalmente, nombre de dominio.

La instalación de un broker MQTT es muy sencilla (una de sus principales ventajas). Para ello basta con instalar el demonio mosquitto a través del siguiente comando:

Ejemplo de instalación en Ubuntu:

```
$ sudo apt-get install mosquitto
```

Por defecto, Ubuntu arranca el servicio después de instalarlo. Si la distribución Lunix sobre la cual se instala no realiza esta acción por defecto, se deberá configurar debidamente para arrancarlo.

En este proyecto se utiliza la configuración por defecto de Mosquitto. Para más información sobre su configuración, consultar manual de instalación de mosquitto.

8.3. Servidor de predicción

8.3.1. Requisitos previos

- Máquina con distribución Linux.
- Python 2.7.
- Conexión a internet.
- IP fija y opcionalmente, nombre de dominio.

8.3.2. Instalación

Los archivos necesarios para la instalación del software del servidor de cálculo se encuentran disponibles en Github.

- Introduce el directorio "server" en el servidor.
- Cambia las siguientes variables en "main.py":
  - brokerIP: IP o nombre de dominio del broker MQTT.
  - brokerPort: Puerto del broker MQTT.
  - topic: Canal del broker MQTT al que escuchar.
  - thingspeakKey: Clave privada del usuario ThingSpeak.

Capítulo 9

Conclusiones

9.1. Estado del proyecto

Los hitos que hemos conseguido desarrollar con éxito son:

- Montar un sistema de recogida de datos. Un nodo muy sencillo de replicar y que fácilmente podría trabajar en una red de estos. Gracias al protocolo MQTT que permite muchos "publicadores" y a que a la hora del entrenamiento se ha tenido en cuenta el identificador del nodo con el que se ha trabajado, se podría añadir al estudio los datos recogidos por múltiples nodos a lo largo de una extensión de terreno.

- Sistema de entrenamiento de datos. El software desarrollado para la obtención y tratamiento de los datos para el entrenamiento permite obtener conjuntos de datos de forma parametrizada. De esta forma, en una posible continuación del proyecto, sería muy sencillo generar sus propios conjuntos de datos para el entrenamiento. Además se han explorado ciertos modelos predictivos que, aún sin mucho éxito, no haría falta volver a estudiar con nuestros parámetros.

- Sistema de predicción. Aunque no se haya conseguido un modelo de predicción óptimo, sí se ha desarrollado el sistema que recibe los datos recogidos y utiliza el modelo elegido para realizar una predicción. Además, debido a la modularidad de nuestro sistema, es muy sencillo implementar nuevos modelos de predicción.

Cada uno de los componentes del sistema se ha desarrollado pensando en una posible futura sustitución o mejora. Asi, cada componente es autónomo y fácilmente reemplazable por otro que respete la interfaz de comunicación. Esto permite que las posibles carencias puedan ser solventadas sin la necesidad de alterar el resto de componentes.

Este proyecto deja un margen de mejora sobre los requisitos iniciales:

- Modelo predictivo. No se ha logrado obtener una predicción suficiente como para poder llevar este sistema a un entorno real. Una de las posibles mejoras sería estudiar otros modelos y parámetros.

- Sistema de visualización. Debido a la falta de tiempo para tomar muestras del nodo, no se ha podido pulir este componente. La información que muestra no aparece de una forma clara. Sin embargo, sí recibe y almacena los datos necesarios y en un futuro sólo sería necesario corregir el fragmento de código que transforma los datos crudos en gráficas representativas.

9.2. Posibles ampliaciones

Desde un primer momento, se tenía como una posible ampliación formar una red de nodos para la recogida de muestras. Se ha empezado por desarrollar uno para no ampliar la complejidad del proyecto.

Se intuye que dentro del modelo predictivo, podría ser interesante disponer de muestras desde distintos puntos de un territorio. Quizá esto podría permitir inferir al modelo predictivo cambios en las condiciones meteorológicas en un nodo a partir de las recogidas por otro.

Por último, otra posible ampliación, sería estudiar el modelo de predicción ARIMA. Que hace uso de regresión lineal teniendo en cuenta valores pasados.

Capítulo 10

Conclusions

10.1. Project status

The milestones we have successfully developed are:

- Set up a data collection system. A very simple node to replicate and it could easily work on a network of these. Thanks to the MQTT protocol that allows many publishers since at the time of the training, the identifier of the node with which it was worked was taken into account, the data collected by multiple nodes could be added to the study. An extension of land.

- Data training system. The software developed for obtaining and processing data for training allows to obtain data sets in a parameterized way. In this way, in a possible continuation of the project, it would be very easy to generate their own data sets for training. In addition, we have explored certain predictive models that, even without much success, it would not need to re-study with our parameters.

- Prediction system. Although an optimal prediction model has not been achieved, the system that receives the collected data has been developed and uses the model chosen to make a prediction. In addition, due to the modularity of our system, it is very easy to implement new prediction models.

Each of the components of the system has been developed with a possible future re-

placement or improvement. Thus, each component is autonomous and easily replaceable by another that respects the communication interface. This allows that the possible deficiencies can be solved without the necessity to alter the rest of components.

This project leaves a margin of improvement over the initial requirements:

- Predictive model. It has not been possible to obtain a prediction sufficient to be able to take this system to a real environment. One of the possible improvements would be to study other models and parameters.

- Display system. Due to the lack of time to sample the node, this component could not be polished. The information shown does not appear clearly. However, it receives and stores the necessary data and in the future it would only be necessary to correct the piece of code that transforms raw data into representative charts.

10.2. Possible extensions

From the outset, it was possible to form a network of nodes for the collection of samples. It has begun to develop one so as not to expand the complexity of the project.

It is intuited that within the predictive model, it could be interesting to have samples from different points of a territory. Perhaps, this could allow to infer to the predictive model changes in the meteorological conditions in a node from those collected by another.

Finally, another possible extension would be to study the prediction model ARIMA. It makes use of linear regression taking into account past values.

Capítulo 11

Trabajo del Alumno

11.1. Pablo Aragón Moreno

11.1.1. Investigación

Tras la primera reunión con los directores acerca del proyecto, el alumno Pablo Aragón Moreno se asignó la investigación de la recogida y tratamiento de los datos, tanto de InfoRiego, como la de los datos meteorológicos del exterior.

Investigó distintas tecnologías para dicho cometido: Ruby, Python, C... Para ello estudió estas posibilidades mediante su documentación, experiencia y opiniones en Internet.

Ya que el algoritmo de predicción y el entrenamiento se iba a realizar en Python, Pablo Aragón Moreno decidió ser consistente con el proyecto y eligió dicho lenguaje. Además, Python es un lenguaje muy potente en lo que al WebScrapping se refiere, dado que tiene muchas librerías de tratamiento y obtención de grandes cantidades de datos.

Pablo Aragón Moreno siguió con su investigación para la recogida de datos meteorológicos. La idea era desarrollar un módulo en la Raspberry que controle y orqueste la obtención de datos mediante dos sensores:

- DHT22: Sensor de temperatura y humedad.
- SP-212: Piranómetro.
- ADC MCP3008: Conversor analógico digital.

El alumno estudió los "datasheets" y documentación de los sensores para decantarse por una tecnología.

Debido a que los fabricantes de los sensores proporcionan librerías en Python para el fácil y cómodo manejo de los dispositivos, y debido a la predisposición de Pablo a elegir dicho lenguaje, la elección de la tecnología en este caso fue muy sencilla.

11.1.2. Desarrollo

Pablo Aragón Moreno comenzó a desarrollar los scripts en Python de recogida de datos de la web InfoRiego.

El trabajo consistía en la recopilación de los datos del servidor de InfoRiego, después descomprimirlos al formato original (CSV) y el tratamiento y clasificación en otros CSVs para así poder normalizar y generar los conjuntos de datos para el "Machine Learning". Esta labor facilitó mucho la tarea de Abel Coronado López para entrenar dichos datos y, más adelante, predecir.

Este desarrollo del tratamiento de datos permitió que se pudiesen formar conjuntos de datos para el entrenamiento de forma parametrizada. Permite indicar qué intervalo de tiempo se requiere y el algoritmo se encarga de comprobar si ya está descargado y en caso contrario lo descarga y posteriormente generar el resto de conjuntos de datos a partir de este que se requieran para el entrenamiento también de forma parametrizada.

Conseguido ese hito, el alumno implementó los algoritmos de recogida de datos meteorológicos del nodo. Para ello se valió de las librerías mencionadas anteriormente.

11.1.3. Documentación

Durante su desarrollo, Pablo Aragón Moreno fue documentando todo su trabajo, tanto en Drive, como en Github.

- En Drive documentó todo aquello relacionado con la tecnología usada (links, trabajos...).

- En Github explicó cómo se instala el nodo: cableado, scripts de configuración... Entre otras cosas.

La última fase del proyecto consistió en documentar toda su parte, tanto de investigación como de desarrollo, en la memoria final. A parte, ayudó a sus compañeros a desarrollar otras secciones...

11.2. María Castañeda López

11.2.1. Investigación

Al comenzar el proyecto, María Castañeda López se estuvo informando de algunos protocolos para poder recoger, almacenar y visualizar los datos que provinieran del nodo (CoAP, MQTT o AMQP). En especial investigó MQTT, muy utilizado en la comunicación máquina a máquina, muy simple. Aunque María Castañeda López y sus compañeros se informaron sobre otras alternativas, MQTT y el servidor Mosquitto fue el elegido, principalmente porque este fue el recomendado por sus tutores.

Otra parte importante que María Castañeda López tuvo que investigar al principio del proyecto, fue la manera en la que el nodo se comunicaba con MQTT. Como Raspberry 3 incluye un módulo de Wi-Fi, esta fue la tecnología que decidió utilizar. Se estudiaron otras alternativas como Lora, la cual utiliza menos recursos y es la más utilizada en sistemas "machine-to-machine", como nuestro proyecto. Al ser sólo un prototipo, decidieron no preocuparse por la optimización de recursos y optar por aprovechar el modulo Wi-Fi nativo en la Raspberry.

La alumna implementó la otra parte de la conexión MQTT, el servidor de predicción. Utiliando la funcionalidad del cliente MQTT implementado en el nodo, lo acopló al sistema del servidor.

Respecto al sistema de visualización, María Castañeda López estudió los sistemas disponibles en este momento en el mercado, como por ejemplo: FreeBoard o IoTDataViz o, incluso, uno desarrollado por ellos mismos. Pero la alumna y sus compañeros decidieron

una vez más, hacer caso a las recomendaciones de sus tutores y utilizar ThinkSpeak. Esta herramienta, además de ser gratuita, te permite la facilidad de manipular los datos de una manera más personalizada. Para comunicarse con ThingSpeak, se decidió desarrollar un módulo denominado "Bridge" para poder mandar los resultados de nuestra predicción al sistema de visualización. Este Bridge aprovecha la API HTTP que proporciona ThingSpeak para realizar la comunicación

11.2.2. Desarrollo

Una vez que María Castañeda López terminó el trabajo de investigación, ella y sus compañeros desarrollaron una serie de scripts relacionados con la comunicación entre los componentes del sistema.

Uno de estos scripts se encarga de la comunicación MQTT del nodo con el servidor de datos mediante la ayuda de la librería paho-mqtt, que facilitó mucho las cosas.

Además María Castañeda López y sus compañeros decidieron crear tres logs, dos en el nodo y otro en el servidor de predicción, para, tener así, un método de registro de los eventos futuros.

En los logs del nodo se almacenan datos recogidos por el nodo en forma de registros diarios para prevenir su pérdida. Como el cliente MQTT está escuchando continuamente, en cuanto recibe los datos, los preservará en dicho log. También almacena el resultado de cada conexión con el broker permitiendo así conocer si la conexión falló en algún momento.

Por otro lado, el log del servidor de predicción guarda todos los datos que recibe para poder utilizarlos en la predicción.

11.2.3. Documentación

La alumna María Castañeda López, fue recopilando documentación sobre los protocolos y tecnologías investigadas para poder desarrollar su trabajo durante todo el proyecto. Y, al igual que el resto de sus compañeros, almacenó dicha información en una carpeta compartida por el equipo de desarrollo del proyecto y en Github. De está manera podían tener tanto

ella, como el resto de sus compañeros, la documentación, problemas, comentarios o dudas de una manera fácil y directa.

María Castañeda López se ha encargado de redactar en este documento todo lo relacionado con su parte de la investigación y del desarrollo, además de aportar su ayuda junto a la del resto de sus compañeros en otros capítulos de este documento.

11.3. Abel Coronado López

11.3.1. Investigación

Durante la fase inicial del proyecto, Abel Coronado López se encargó de la investigación de las distintas tecnologías para desarrollar "Machine Learning" que se encontraban actualmente en el "mercado" y seleccionar cuál era la que mejor le venía al proyecto, y cuáles eran los motivos por los que había escogido la tecnología.

En un principio, la tecnología a utilizar sería MATLAB, pero, después de la investigación previa, se descubrió que Python también tiene una gran potencia en este sector. Python cuenta con un gran abanico de librerías para trabajar con grandes cantidades de datos, análisis científico, inteligencia artificial... Cuenta también con distribuciones que ya incluyen gran cantidad de estas librerías para poder hacer uso de ellas sin necesidad de instalarlas una a una como "Anaconda". Además, junto con la facilidad de programar en este lenguaje y la experiencia previa de los componentes del grupo con él, hizo que esta fuera la elección final.

Tras tener elegida la tecnología con la que desarrollar el proyecto, Abel Coronado López, investigó distintos algoritmos matemáticos para el "Machine Learning" y cómo poder implementarlos.

Hizo uso de cursos online, como por ejemplo Udacity, y se valió de tutoriales básicos de "Machine Learning" en YouTube para coger una idea básica de este nuevo paradigma emergente.

Con una idea más clara de los objetivos del proyecto y de cómo implementarlos, el

alumno estudió pequeños problemas resueltos relacionados con la IA propuestos en Intenet:

- Desastre del Titanic
- Investigación de la diabetes
- Predicción del divorcio

Cuando la fase de estudio de los algoritmos concluyó, el alumno, junto al resto de componentes del equipo, generó una lista de posibles algoritmos para utilizar en el proyecto:

- Regresión Lineal
- SVM
- Clasificador

Esta lista iría cambiando según avanzara el proyecto debido a diversos factores:

- Poca documentación
- Problemas de implementación
- Malos resultados

En esta fase de investigación, se descubrió un método de estudio para automatizar las pruebas y hacer más viable la experimentación de modelos y parámetros usados en el "Machine Learning": Grid Search 5.3.

11.3.2. Desarrollo

Con la fase de investigación finalizada, Abel Coronado López, junto con el resto de compañeros, empezaron con el desarrollo e implementación de estos algoritmos en el proyecto.

Gracias a Grid Seach este trabajo fue mucho más viable y cómodo.

Se desarrolló una plantilla que usa Grid Search para abstraer los modelos y parámetros a explorar. De esta manera, la fase de entrenamiento sería configurable mediante archivos "JSON". El alumno desarrolló dichos ficheros explicados previamente. Ver figura 5.10.

Una vez finalizada la fase de entrenamiento con distintos modelos y configuraciones, el equipo eligió el algoritmo que mejor precisión de predicción tiene para nuestro contexto.

11.3.3. Documentación

Durante todos los procesos anteriormente descritos, el alumno Abel Coronado López documentó todo lo relacionado con las tecnologías investigadas y utilizadas por el sistema en una carpeta compartida por el equipo de desarrollo del proyecto, y en Github. Esta documentación consistía en comentar todo lo que se iba desarrollando e investigando, de forma que los demás compañeros, si querían introducirse en esa parte del sistema, únicamente tendrían que leer dónde y cómo se hizo.

En la fase final del proyecto, el alumno ha documentado todo el sistema relacionado con la predicción para que este fuese introducido en la memoria del proyecto, también a aportado en otras secciones como la introducción, el nodo, etc...

Bibliografía

[1] Adafruit. https://cdn-shop.adafruit.com. Cdn-shop.adafruit.com. N. p., 2017. Web.

[2] Apogee instruments. https://www.apogeeinstruments.co.uk/content/SP-212-215-manual.pdf. Apogeeinstruments.co.uk. N. p., 2017. Web.

[3] Codacy. http://www.codacy.com. ."Automated Code Reviews and Code Analytics. Codacy". Codacy.com. N. p., 2017. Web.

[4] Numpy. http://www.numpy.org. "Numpy — Numpy". Numpy.org. N. p., 2017. Web.

[5] Paho. http://www.eclipse.org/paho. ."Eclipse.org/paho". Eclipse.org. N. p., 2017. Web.

[6] Python. http://www.python.org. "Welcome To Python.Org". Python.org. N. p., 2017. Web.

[7] Raspberry pi. https://www.raspberrypi.org. Teach, Learn, and Make with Raspberry Pi.

[8] Scikit-learn. www.scikit-learn.org. "Scikit-Learn: Machine Learning In Python — Scikit-Learn 0.18.1 Documentation". Scikit-learn.org. N. p., 2017. Web.

[9] Wiring pi. http://www.wiringpi.com. "Wiringpi". Wiringpi.com. N. p., 2017. Web.

Capítulo 12

Anexo: Gráficas Regresión Lineal

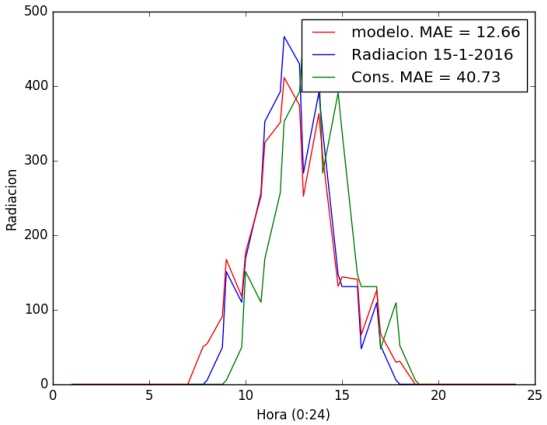

Figura 12.1: *Resultado 15-01-2016*

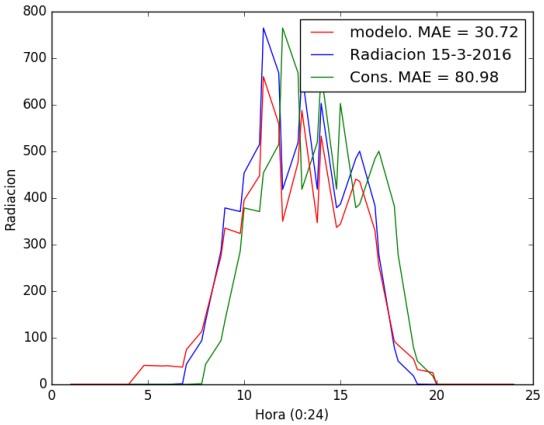

Figura 12.2: *Resultado 15-03-2016*

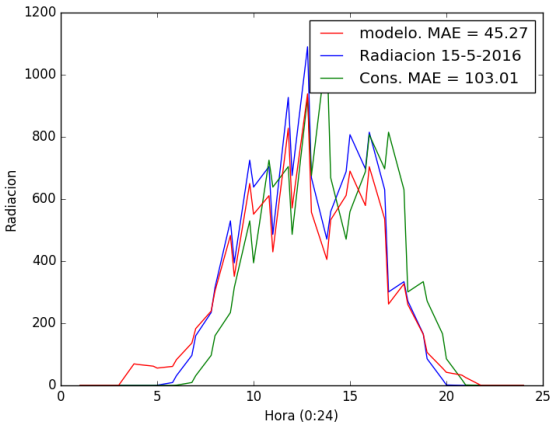

Figura 12.3: *Resultado 15-05-2016*

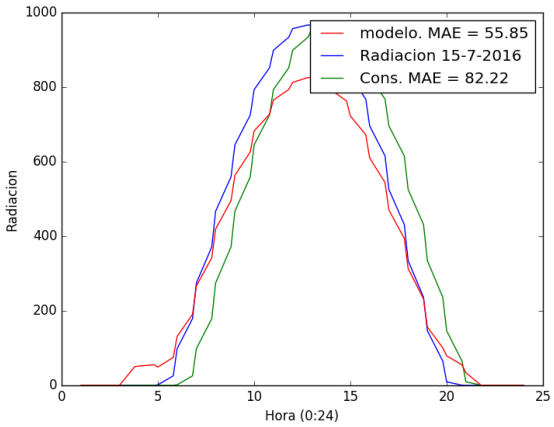

Figura 12.4: *Resultado 15-07-2016*

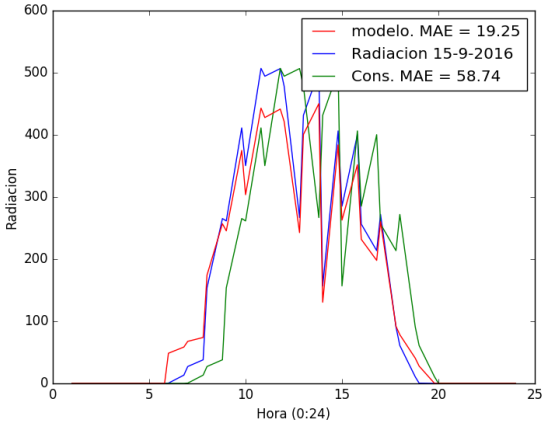

Figura 12.5: *Resultado 15-09-2016*

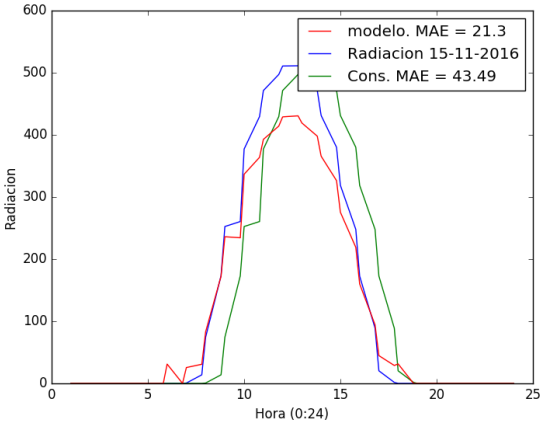

Figura 12.6: *Resultado 15-11-2016*

Capítulo 13

Anexo: Gráficas SVR

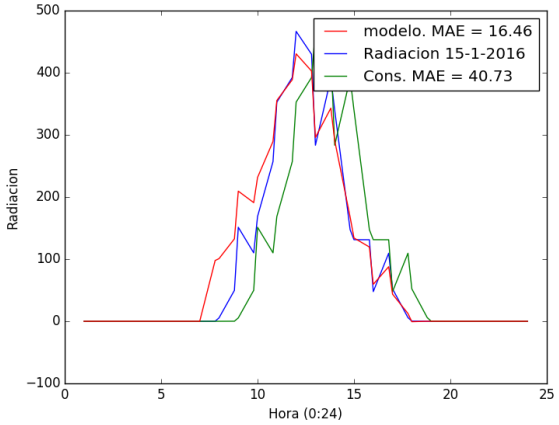

Figura 13.1: *Resultado 15-01-2016*

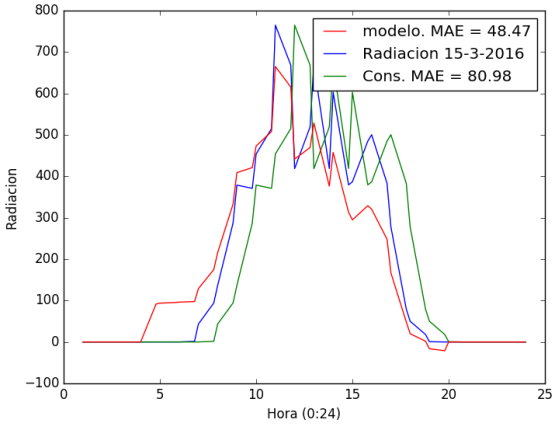

Figura 13.2: *Resultado 15-03-2016*

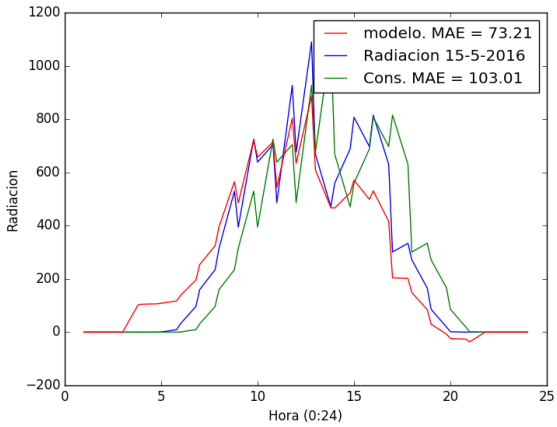

Figura 13.3: *Resultado 15-05-2016*

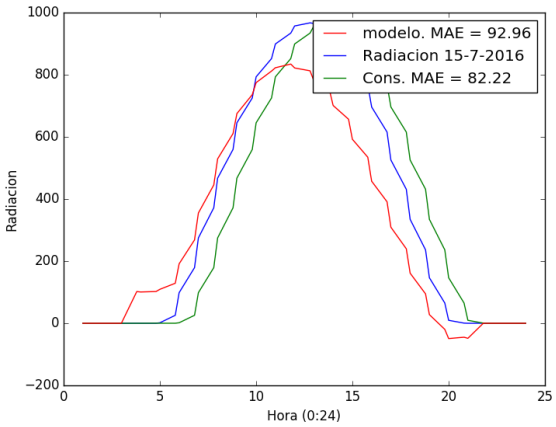

Figura 13.4: *Resultado 15-07-2016*

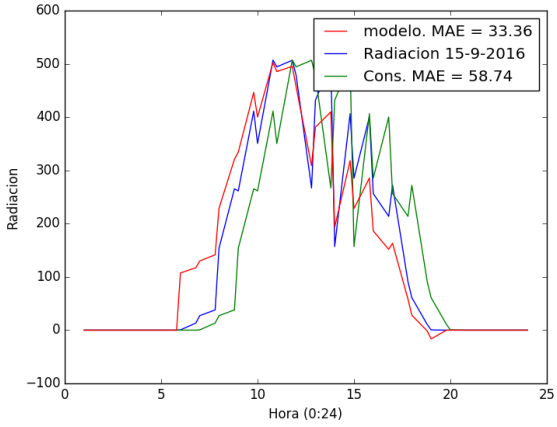

Figura 13.5: *Resultado 15-09-2016*

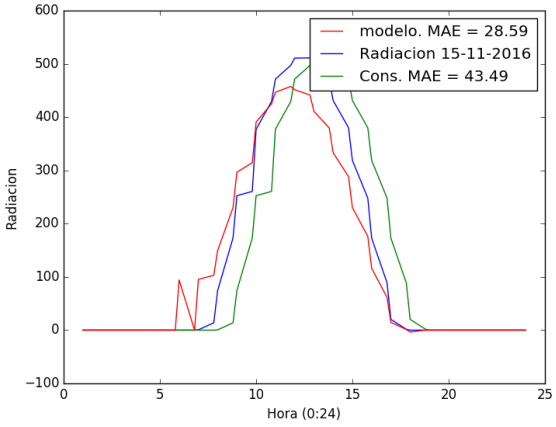

Figura 13.6: *Resultado 15-11-2016*

Capítulo 14

Anexo: Gráficas Redes Neuronales

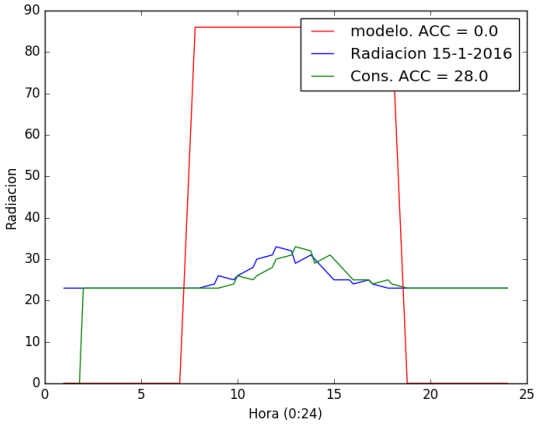

Figura 14.1: *Resultado 15-01-2016*

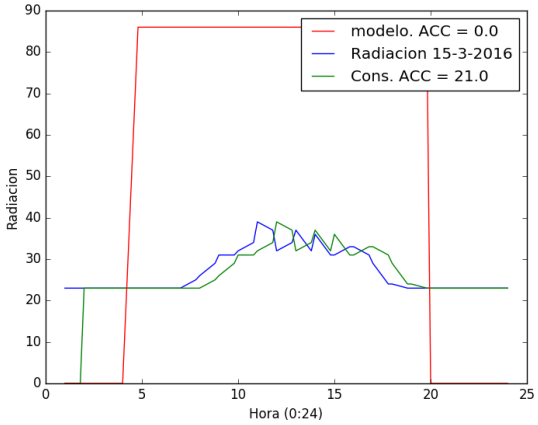

Figura 14.2: *Resultado 15-03-2016*

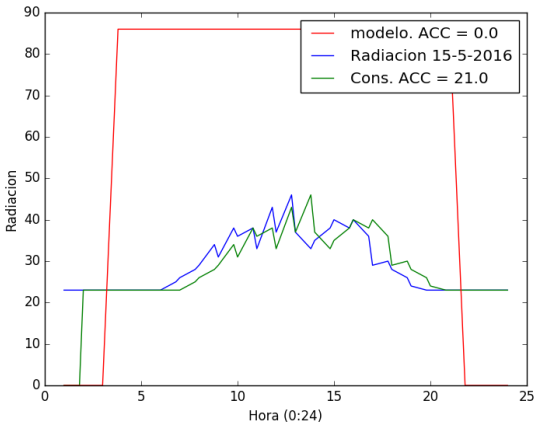

Figura 14.3: *Resultado 15-05-2016*

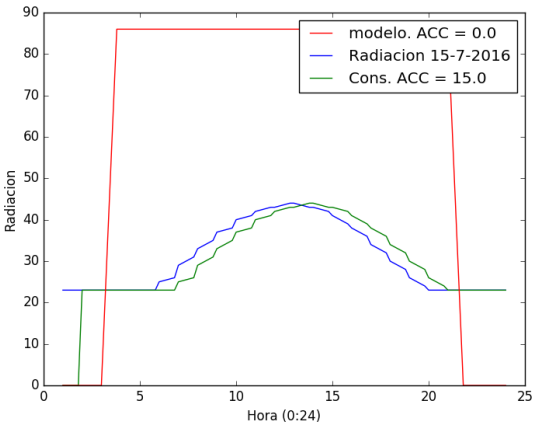

Figura 14.4: *Resultado 15-07-2016*

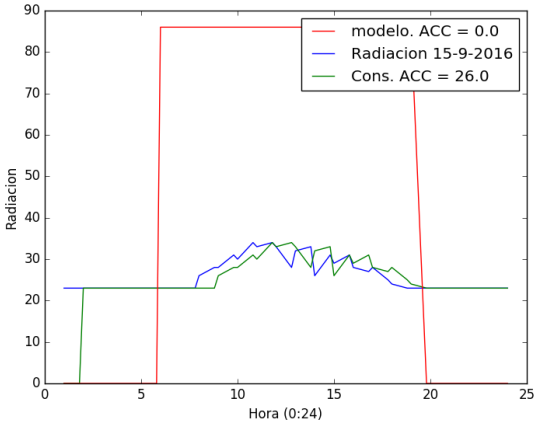

Figura 14.5: *Resultado 15-09-2016*

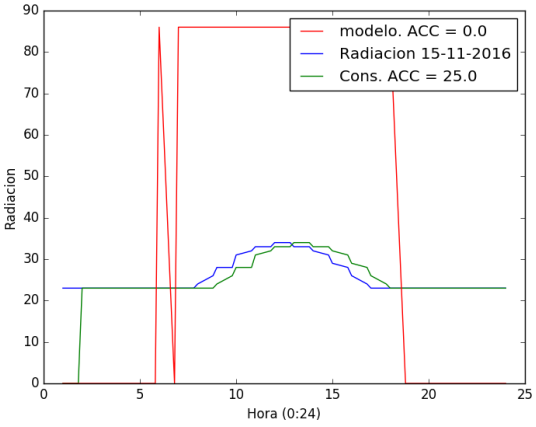

Figura 14.6: *Resultado 15-11-2016*

Capítulo 15

Anexo: Calidad del código

Una vez acabado el código del proyecto, decidimos certificar su calidad mediante Codacy. Codacy es una herramienta que proporciona nuestro controlador de versiones GitHub. Sencillamente revisa todas y cada una de las líneas del código, para hacerlo más sencillo, escalable y seguro. Genera un informe con los errores y te dice qué grado de calidad tiene, siendo A el más alto. ([3])

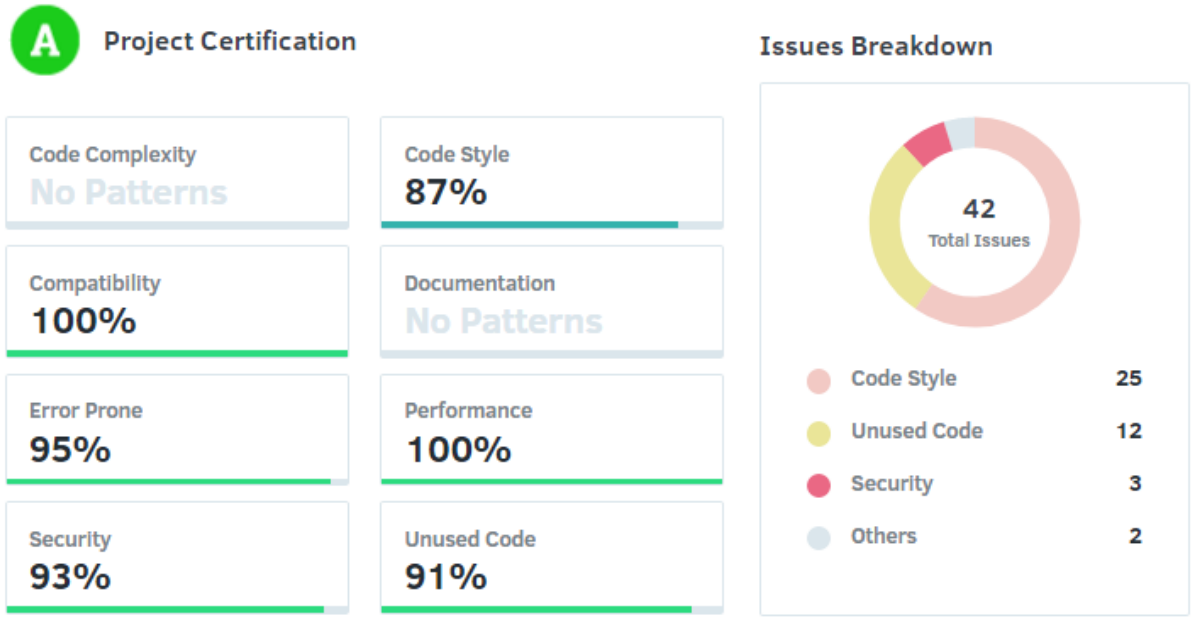

Figura 15.1: *Informe de nuestro proyecto [Fuente: Codacy]*