ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO 201113274

MODELO PREDICTIVO. MACHINE LEARNING APLICADO AL ANÁLISIS DE DATOS CLIMÁTICOS CAPTURADOS POR UNA PLACA SPARKFUN

Autor: Diego Iribarren Baró
Director: Jaime Pereña Pinedo

Madrid

Julio de 2016

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN ACCESO *ABIERTO* (*RESTRINGIDO*) DE DOCUMENTACIÓN

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Diego Iribarren Baró, como alumno de la UNIVERSIDAD PONTIFICIA COMILLAS (COMILLAS), **DECLARA**

que es el titular de los derechos de propiedad intelectual, objeto de la presente cesión, en relación con la obra proyecto fin de carrera[1], que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual como titular único o cotitular de la obra.

En caso de ser cotitular, el autor (firmante) declara asimismo que cuenta con el consentimiento de los restantes titulares para hacer la presente cesión. En caso de previa cesión a terceros de derechos de explotación de la obra, el autor declara que tiene la oportuna autorización de dichos titulares de derechos a los fines de esta cesión o bien que retiene la facultad de ceder estos derechos en la forma prevista en la presente cesión y así lo acredita.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad y hacer posible su utilización de *forma libre y gratuita* (*con las limitaciones que más adelante se detallan*) por todos los usuarios del repositorio y del portal e-ciencia, el autor **CEDE** a la Universidad Pontificia Comillas de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución, de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra (a) del apartado siguiente.

3º. Condiciones de la cesión.

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia, el repositorio institucional podrá:

(a) Transformarla para adaptarla a cualquier tecnología susceptible de incorporarla a internet; realizar adaptaciones para hacer posible la utilización de la obra en formatos electrónicos, así

[1] Especificar si es una tesis doctoral, proyecto fin de carrera, proyecto fin de Máster o cualquier otro trabajo que deba ser objeto de evaluación académica

como incorporar metadatos para realizar el registro de la obra e incorporar “marcas de agua” o cualquier otro sistema de seguridad o de protección.

(b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.

(c) Comunicarla y ponerla a disposición del público a través de un archivo abierto institucional, accesible de modo libre y gratuito a través de internet.[2]

(d) Distribuir copias electrónicas de la obra a los usuarios en un soporte digital. [3]

4º. Derechos del autor.

El autor, en tanto que titular de una obra que cede con carácter no exclusivo a la Universidad por medio de su registro en el Repositorio Institucional tiene derecho a:

a) A que la Universidad identifique claramente su nombre como el autor o propietario de los derechos del documento.

b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.

c) Solicitar la retirada de la obra del repositorio por causa justificada. A tal fin deberá ponerse en contacto con el vicerrector/a de investigación (curiarte@rec.upcomillas.es).

d) Autorizar expresamente a COMILLAS para, en su caso, realizar los trámites necesarios para la obtención del ISBN.

d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

---

[2] En el supuesto de que el autor opte por el acceso restringido, este apartado quedaría redactado en los siguientes términos:

(c) Comunicarla y ponerla a disposición del público a través de un archivo institucional, accesible de modo restringido, en los términos previstos en el Reglamento del Repositorio Institucional

[3] En el supuesto de que el autor opte por el acceso restringido, este apartado quedaría eliminado.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.

b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.

c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

a) Deberes del repositorio Institucional:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.

- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusiva del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.

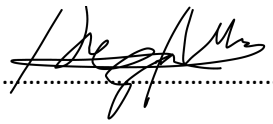
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.

b) Derechos que se reserva el Repositorio institucional respecto de las obras en él registradas:

- retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 20 de Julio de 2016

ACEPTA

Fdo…… 

Proyecto realizado por el alumno/a:

Diego Iribarren Baró

Fdo.: ..... Fecha: 20 / 07 / 2016

Autorizada la entrega del proyecto cuya información no es de carácter confidencial

EL DIRECTOR DEL PROYECTO

Jaime Pereña Pinedo

Fdo.: ..... Fecha: 20 / 07 / 2016

Vº Bº del Coordinador de Proyectos

Álvaro Sánchez Miralles

Fdo.: ..... Fecha: 20 / 07 / 2016

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO 201113274

MODELO PREDICTIVO. MACHINE LEARNING APLICADO AL ANÁLISIS DE DATOS CLIMÁTICOS CAPTURADOS POR UNA PLACA SPARKFUN

Autor: Diego Iribarren Baró
Director: Jaime Pereña Pinedo

Madrid

Julio de 2016

MODELO PREDICTIVO. MACHINE LEARNING APLICADO AL ANÁLISIS DE DATOS CLIMÁTICOS CAPTURADOS POR UNA PLACA SPARKFUN.

Autor: Iribarren Baró, Diego.

Director: Pereña Pinedo, Jaime.

Entidad colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Resumen - El proyecto que se resume a continuación tiene como objetivo principal el desarrollo de un modelo de predicción que determine en tiempo real la probabilidad de cancelación o retraso de un vuelo en función de las condiciones meteorológicas. Para ello, se utiliza una placa Sparkfun como dispositivo de toma de datos meteorológicos, la plataforma en la nube Azure para desarrollar y ejecutar el modelo y PowerBI como herramienta de procesamiento y visualización de datos.

Palabras clave - Sensores, Internet of Things, Big Data, Machine Learning.

1. INTRODUCCIÓN

1.1. Objetivos del proyecto

El fin de este proyecto es crear un modelo capaz de proporcionar una predicción precisa sobre posibles retrasos o cancelaciones de vuelos debidos a las condiciones climáticas. Para ello se han definido los siguientes objetivos:

- Captar con exactitud las condiciones meteorológicas relevantes para la predicción.
- Transmitir y procesar los datos obtenidos para alimentar con ellos el modelo, configurando correctamente las conexiones necesarias entre el dispositivo y la plataforma en la nube.
- Crear un modelo predictivo en la nube que sea preciso y robusto, y que permita obtener una predicción correcta en la mayoría de los casos.
- Presentar los datos almacenados y los resultados obtenidos de una forma visual y sencilla de comprender.

1.2. Estado del arte y motivación

Desde la creación del primer sensor en 1958 hasta la actualidad, estos dispositivos de medida de parámetros físicos han estado siempre presentes en el mundo de la tecnología [1]. Con el objetivo de mejorar la precisión de las medidas, la frecuencia de toma de datos y el tamaño de los dispositivos, el desarrollo de los sensores los ha convertido en la principal fuente de información que se utiliza actualmente.

La computación en la nube (en inglés 'cloud computing') nació en torno a 1960, pero su verdadero desarrollo comenzó en los años 90 de la mano de Salesforce y Amazon, y más adelante Google y Microsoft entre otros. Este sistema de prestación de servicios cuenta con una serie de características que hacen de él un elemento imprescindible hoy en día: el alto grado de automatización, la capacidad de adaptación y movilización de recursos, la virtualización avanzada, el pago en función del consumo y la elevada seguridad [2].

Figura 1. Computación en la nube, Internet de las cosas.

De forma paralela al cloud computing por estar intrínsecamente relacionados, la

tecnología Big Data ha ido ganando peso progresivamente en prácticamente todos los campos de negocio. La capacidad de almacenar y analizar inmensas cantidades de datos tiene infinidad de aplicaciones con grandes beneficios para las compañías y usuarios.

Figura 2. Big Data.

El nexo de unión de todos los elementos ya mencionados es el Internet de las Cosas (en inglés 'IoT'). El origen del IoT se sitúa en 1999, cuando Bill Joy y Kevin Ashton hablaron por primera vez de una comunicación dispositivo-dispositivo. Este sistema consiste en la conexión de cualquier dispositivo a la nube mediante etiquetas de radio de baja frecuencia (RFID), de manera que se puedan obtener grandes cantidades de información y gestionarlas de forma automatizada [3]. Es, en definitiva, una manera de transferir automáticamente los datos medidos por sensores a la nube, para poder almacenarlos y analizarlos en tiempo real. La combinación de las nuevas y desarrolladas características de sensores, cloud computing y Big Data, abre un campo totalmente nuevo en cuanto a cantidad de información disponible para analizar y recursos para hacerlo de forma rápida, sencilla y efectiva.

En los últimos años se han aplicado estos avances en numerosos proyectos de todo tipo y en todas las industrias. La estructura de la mayoría de ellos se compone de la toma de datos mediante dispositivos, la transmisión de esos datos a la nube, el uso de Big Data para procesar o analizar esos datos y la obtención de resultados aplicables a la resolución de un problema o la mejora del negocio.

La principal característica de este proyecto es la integración de todos los recursos implicados en este proceso, desde la configuración del dispositivo al más bajo nivel hasta el análisis de la información obtenida después del tratamiento y procesamiento de los datos. Es por tanto un proyecto 'end-to-end' que cubre todo el abanico de tecnologías implicadas en la creación, configuración y aplicación de un modelo predictivo.

1.3. Recursos utilizados

Los recursos que se han utilizado en el desarrollo del proyecto se pueden clasificar en tres tipos: hardware, software y datos.

Hardware

La placa Sparkfun y el accesorio Weather Shield son los dos elementos utilizados para la toma de datos meteorológicos en tiempo real.

Figura 3. Placa Sparkfun y Weather Shield.

Estos dos dispositivos incluyen una serie de sensores que permiten captar las condiciones meteorológicas con una frecuencia del orden de milisegundos y enviarlas en tiempo real a un ordenador para analizarlas o almacenarlas. Para este proyecto se han registrado las medidas de temperatura, humedad relativa y presión.

Software

La plataforma en la nube de Microsoft, Azure, es el entorno en el que se ha desarrollado la mayor parte del proyecto.

Figura 4. Portal de Azure, infraestructura en la nube.

Esta infraestructura pone a disposición del usuario una gran cantidad de servicios con infinidad de aplicaciones. En este caso, se

han utilizado los recursos relacionados con IoT y Machine Learning, aprovechando también la gran capacidad de tratamiento de Big Data de que se dispone.
Los datos capturados con la placa han sido enviados a Azure para su procesamiento y almacenamiento, y el modelo predictivo está creado y ejecutado totalmente en la nube.

Datos

Para entrenar el modelo y poder elegir el algoritmo más adecuado con el objetivo de obtener predicciones más precisas, se han utilizado dos conjuntos de datos obtenidas de AENA y AEMET. Estos dos conjuntos contienen información de los vuelos con origen en el aeropuerto de Madrid Barajas y las condiciones meteorológicas en la misma zona durante los últimos tres años (de 2013 a 2015) respectivamente.

2. METODOLOGÍA

2.1. Configuración de la placa

En primer lugar, se realiza la configuración básica de la placa Sparkfun utilizando el programa Arduino. Se instala el IDE y los drivers necesarios para la conexión USB del dispositivo al ordenador, y se actualiza la Firmata.
Posteriormente, se escribe el código necesario para la toma de datos, un Javascript que determina los parámetros a medir, los nombres y tipos de variable donde almacenarlos y la frecuencia de medida.

2.2. Configuración de la nube

A continuación, se procede a la configuración del entorno de trabajo en la nube, creando en primer lugar un IoT Hub. Este servicio se distingue de otros similares ya que permite la conexión y gestión de uno o varios dispositivos a la nube, centrándose especialmente en la comunicación del dispositivo a la nube y de la nube al dispositivo, así como en la monitorización de las conexiones mediante el registro de eventos [4].

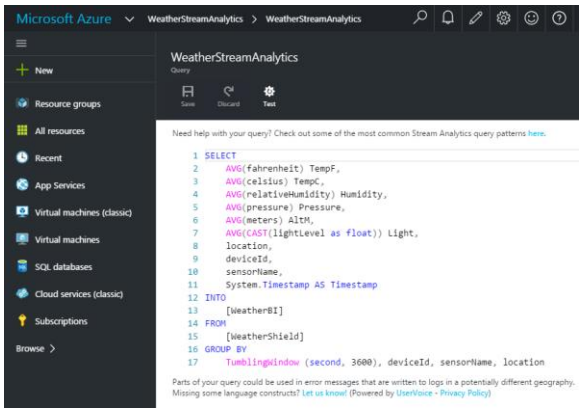


Figura 5. Query de Stream Analytics.

Además, se utiliza Stream Analytics como herramienta de transmisión de datos, haciendo uso también de la capacidad de crear una consulta o 'query' para consolidar a cada hora los datos recibidos.

2.3. Desarrollo del modelo

Azure dispone de una interfaz dedicada exclusivamente a proyectos o experimentos de Machine Learning, el Machine Learning Studio. La creación y desarrollo del modelo se realiza en este entorno, y como en la mayoría de proyectos de Machine Learning se pueden distinguir tres fases:
Preparación de los datos: en este primer paso, que puede constar de varias etapas iterativas, se combinan todos los datos meteorológicos en un único archivo y se modifica su formato para hacerlo coincidir con el histórico de vuelos. Se obtienen dos ficheros .csv, uno consolidando el registro de parámetros meteorológicos y otro con el registro de todos los vuelos salidos desde Barajas.
Elección del algoritmo: para este proyecto se decide usar algoritmos de clasificación multiclase, que determinan la categoría en la que se encuentra cada caso entre 'En hora', 'Retrasado' o 'Cancelado'. Se crea un experimento en Machine Learning Studio dedicado a comparar los resultados de los algoritmos más destacados [5]. Inicialmente fueron 'multiclass decision forest', 'multiclass decision jungle', 'multiclass logistic regression' y 'multiclass neural network'.

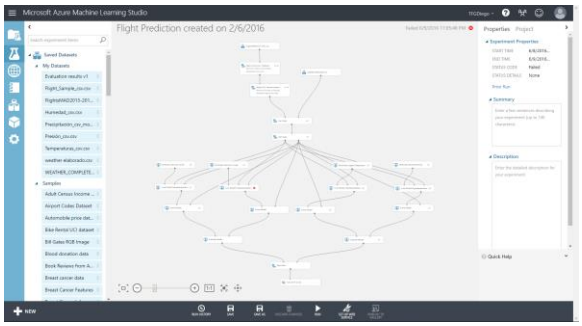


Figura 6.Elección del algoritmo, Machine Learning Studio.

Esta segunda fase permite comparar la precisión, porcentaje de acierto en cada una de las categorías y tiempo de ejecución del modelo de cada algoritmo.

Despliegue del modelo: una vez elegido el algoritmo más preciso, se procede a crear el modelo predictivo, incluyendo una fase de entrenamiento del modelo con los propios datos de que se disponía. Por último, se despliega el modelo implementando un 'Web Service', que permite acceder al modelo como un recurso más de la web, similar a una applet. Además, se configura una hoja Excel como entrada de la consulta y salida de los resultados de la predicción, para poder acceder a los datos y a la predicción en un entorno más familiar y sencillo.

3. RESULTADOS

3.1. Elección del algoritmo

Después de ejecutar el modelo creado para la elección del algoritmo más preciso, los resultados son los siguientes:

Multiclass decision jungle: la parte del modelo que evaluaba este algoritmo falló, por lo que no se ha tenido en cuenta como opción.

Multiclass logistic regression: presenta un 85,65% de precisión media, pero el acierto por clases en los vuelos cancelados y retrasados es muy bajo.

Multiclass neural network: presenta un 85,66% de precisión media, pero el acierto por clases en los vuelos cancelados y retrasados sigue siendo bajo.

Multiclass decision forest: presenta un 85,82% de precisión media, y es el algoritmo más acertado a la hora de predecir retraso o cancelación.

El algoritmo elegido para el modelo final fue el último, ya que es el más acertado a la hora de predecir retraso y cancelación de los vuelos. Dichos resultados se comentan en el siguiente apartado.

3.2. Resultados de la predicción

Los resultados obtenidos tras la ejecución del modelo utilizando el algoritmo 'Multiclass decision forest' se presentan a continuación.

En primer lugar, la herramienta de evaluación de modelos de Machine Learning Studio proporciona una serie de parámetros que miden el rendimiento del modelo. Estos parámetros se encuentran en la figura 7.

Overall accuracy	0.787319
Average accuracy	0.858213
Micro-averaged precision	0.787319
Macro-averaged precision	0.797411
Micro-averaged recall	0.787319
Macro-averaged recall	0.365375

Figura 7. Métricas de evaluación del modelo predictivo.

Además, se dispone de otra forma más visual de evaluación, conocida como matriz de confusión. Con este método se puede apreciar fácilmente cuál es la respuesta del modelo a cada uno de los tres casos posibles ('En hora', 'Retrasado' o 'Cancelado'). La matriz de confusión del modelo se presenta en la figura 8.

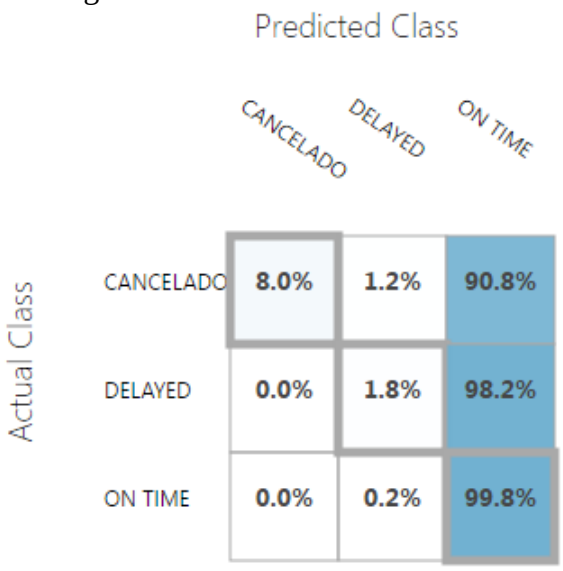


Figura 8. Matriz de confusión del modelo predictivo.

Las conclusiones extraídas a partir de los resultados se comentan en el siguiente apartado.

4. CONCLUSIONES

Para finalizar, a modo de conclusión se realiza un análisis de las tareas y objetivos cumplidos durante el desarrollo del proyecto, así como una interpretación de los resultados obtenidos y posibles mejoras a implementar en el futuro.

Las tareas y objetivos cuya consecución se ha logrado durante el proyecto son las siguientes:

- Se ha logrado con éxito captar datos meteorológicos en tiempo real, cumpliendo así el primer objetivo del proyecto. La frecuencia de captación de datos y la precisión de las medidas han sido excelentes.
- Se ha logrado con éxito configurar correctamente la interfaz en la nube y la conexión del dispositivo con dicho entorno. La transmisión de datos dispositivo-nube ha sido satisfactoria, y además se ha integrado la consolidación en la nube de los datos recibidos.
- Se ha logrado con éxito crear un modelo predictivo en la nube, completando todas las etapas habituales en este tipo de proyectos (tratamiento de los datos, elección del algoritmo y despliegue del modelo). Se ha comprobado la gran complejidad del modelo, y se han determinado posibles mejoras a implementar para aumentar la precisión de los resultados a la hora de realizar la predicción.
- Se ha logrado con éxito realizar un proyecto 'end-to-end', cubriendo con suficiente detalle una gran variedad de tecnologías y recursos desde la captura inicial de los datos hasta la obtención y análisis de los resultados de la predicción.

A continuación, se presentan las conclusiones extraídas a partir del análisis de los resultados:

- Queda patente la gran complejidad de este tipo de modelos predictivos, no sólo por la cantidad de opciones a configurar durante la creación del mismo, sino también por la necesidad de corrección de errores durante el proceso y los tiempos de ejecución del modelo.
- A la vista de los resultados, las condiciones meteorológicas son sólo uno de los factores que pueden causar retrasos y cancelaciones en los vuelos, y esta es la principal causa de que el porcentaje de acierto en estos casos sea menor.

Por último, se presentan algunas de las posibles soluciones a considerar en el futuro:

- Para hacer la predicción más precisa, sería necesario disponer de información relativa a otras causas de cancelación y retraso, como por ejemplo fallos de gestión, retrasos de pasajeros, retrasos en vuelos anteriores, disponibilidad de pistas, fallos técnicos en las aeronaves, motivos burocráticos o políticos y muchos otros. La principal ventaja de este modelo es la fácil implementación de nuevos datos, tras una simple fase inicial de pretratamiento y formato.
- Otra de las opciones a considerar en el futuro para mejorar la experiencia del usuario (destinatario final de la predicción), sería la creación de una aplicación móvil que implementase el modelo y presentase los resultados de una forma sencilla.

5. REFERENCIAS

[1] https://es.wikipedia.org/wiki/Sensor

[2] https://azure.microsoft.com/es-es/overview/what-is-azure/

[3] https://es.wikipedia.org/wiki/Internet_de_las_cosas

[4] https://azure.microsoft.com/es-es/services/iot-hub/

[5] https://azure.microsoft.com/es-es/documentation/articles/machine-learning-algorithm-choice/

PREDICTIVE MODEL. MACHINE LEARNING APPLIED TO ANALYSIS OF WEATHER DATA CAPTURED WITH A SPARKFUN BOARD.

Author: Iribarren Baró, Diego.

Director: Pereña Pinedo, Jaime.

Partner Entity: ICAI – Universidad Pontificia Comillas

PROJECT SUMMARY

Summary – The main objective of this project is creating a real-time prediction model that determines the probability of a flight being cancelled or delayed because of weather conditions. To achieve this goal, a Sparkfun board acts as the data capturing device, the cloud platform 'Azure' is used to develop and execute the model and PowerBI is the tool for data processing and data visualization.

Index terms - Sensors, Internet of Things, Big Data, Machine Learning.

1. INTRODUCTION

1.1. Project goals

The main goal of this project is to create a model capable of making a reliable prediction about possible flight delays or cancellations due to weather conditions. To do so, the following objectives have been defined:

- Capture accurately the relevant weather conditions for making the prediction.
- Transmit and process the obtained data to feed the model with them, configuring correctly the necessary connections between the device and the cloud platform.
- Create a predictive model running in the cloud that will be accurate and robust, and will provide a correct prediction in most of the cases.
- Present the stored data and the obtained results in a visual and easy to understand way.

1.2. State of the art and motivation

Since the creation of the first sensor in 1958 until today, these devices for physical parameters measurement have always been present in the technology business [1]. With the goal of improving the accuracy of the measurements, the frequency of data capturing and the size of the devices, the development of sensors has made them become the main source of information used nowadays.

The use of the term 'Cloud computing' initially started around 1960, but its true development began in the 90's, driven especially by Salesforce and Amazon first and Google and Microsoft among others later. This system for service providing includes a series of characteristics that make it a key element today: high degree of automation, adaptability and resource mobilization capabilities, advanced virtualization, pay based on usage and high security [2].

Figure 9. Cloud computing, Internet of Things.

In parallel to cloud computing, as they are intrinsically related, the use of Big Data

technology has progressively become more and more important in almost every business field. The ability to store, process and analyze huge amounts of data has numerous applications providing many great benefits for companies and users.

Figure 10. Big Data.

The link between all the already mentioned elements is Internet of Things (also known as 'IoT'). This technology first started in 1999, when Bill Joy and Kevin Ashton mentioned a specific new way of device-to-device communication. After some development done until nowadays, this system consists of the connection of any device to the cloud through low frequency radio tags (RFID), in a way that huge amounts of information can be obtained and managed in an automated manner [3]. It is, in a simple way, a new technology used to transmit automatically to the cloud the data measured by sensors, in order to store and analyze that data in real time. The combination of the new and developed characteristics of sensors, cloud computing and Big Data opens a new world of possibilities in terms of quantity of available information to analyze and resources to do so in an easy, effective and quick way.

During the last years, this new technologies and improvements have been applied in multiple projects of all kind and through all industries. The structure of most of them includes a data capturing phase done with devices, the transmission of that data to the cloud, the use of Big Data to process or analyze that data and the obtaining of useful results, applicable to solving a problem or improving the business.

The main characteristic of this project is the integration of all the resources used throughout the whole execution of this process, from the lowest-level configuration of the data capturing device to the analysis of the information obtained after formatting, storing and processing of the data. Therefore, it is an end-to-end project, covering each and every technology involved in the creation, configuration, execution and application of a predictive model based in a cloud environment.

1.3. Resources used

The resources used in the development of the project can be classified in three different categories: hardware, software and data.

Hardware

The Sparkfun RedBoard and the Weather Shield are the two elements used for real-time weather data capturing as the first part of this project.

Figure 3. Sparkfun board and Weather Shield.

These two devices include a series of sensors that allow weather conditions capturing with a frequency of milliseconds, and real-time transmission of that data to a laptop for storage or analysis. The parameters measured for this project are temperature, relative humidity and pressure.

Software

Microsoft's cloud platform, Azure, is the environment in which the greatest part of the development of the project has taken place,

Figure 411. Azure portal, cloud infrastructure.

This infrastructure provides the user with many services and applications they can use. In this case, the used resources have been the ones related with IoT and Machine Learning,

taking advantage as well of the Big Data analysis and processing capabilities the cloud has.

Data are captured with the board and sent to Azure for its processing and storage, and the predictive model is created and run fully on cloud.

Data

Two separate datasets were obtained from AENA and AEMET to train the model. This step is key in being able to choose the most appropriate algorithm for making accurate predictions. The two datasets used in the project contain respectively information of all the flights departing from Madrid Barajas airport and the weather conditions in the area during the last three years (from 2013 to 2015).

2. METHODOLOGY

2.1. Board configuration

First, the basic configuration of the Sparkfun board is done using Arduino. The new IDE is installed, and so are the drivers for the device to laptop USB connection. Also, the Firmata is updated.

The next step is creating the code for the data capturing. This code is a Javascript that determines the parameters to be measured, the names and types of the variables where the data will be stored and, most important, the frequency at which the data capturing occurred.

2.2. Cloud configuration

After configuring the Sparkfun board, the next step of the process is setting up the cloud environment. Creating an IoT Hub is the first thing to do. This service allows connection and management of multiple devices to the cloud, focusing on device-to-cloud and cloud-to-device communication, as well as monitoring of the connections through events log [4].

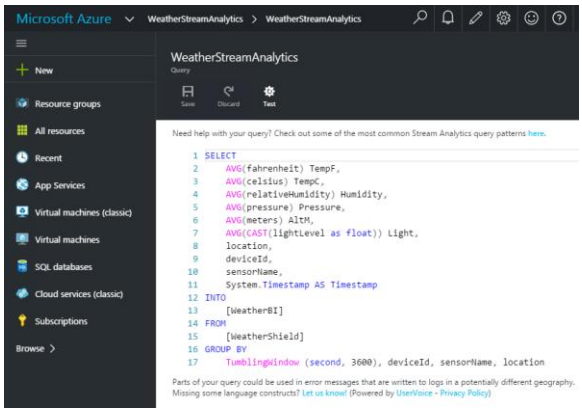


Figure 5. Stream Analytics query.

Also, Stream Analytics is included as a tool for data transmission applying its query creation capability as well. The query created is used for consolidating the received data after every hour.

2.3. Model development

Azure contains an environment devoted exclusively to Machine Learning projects or experiments: Machine Learning Studio. The model is created and developed in this environment entirely, and as in most Machine Learning projects, it can be split into three different phases:

Data preparation: in this first step, that can contain different iterative parts, all the weather data is combined into a single file. The format of the data is modified in order to make it match with the flight historic data. Two .csv files are obtained, one consolidating all the meteorological parameters and another one with the information from all the flights departing from Barajas airport.

Algorithm selection: multiclass classification algorithms are the most appropriate for this project. These algorithms classify each case into three different categories: 'On time', 'Delayed' or 'Cancelled'. The most important algorithms of this type are compared creating an experiment in Machine Learning Studio [5]. Initially, the following are evaluated: 'multiclass decision forest', 'multiclass decision jungle', 'multiclass logistic regression' and 'multiclass neural network'.

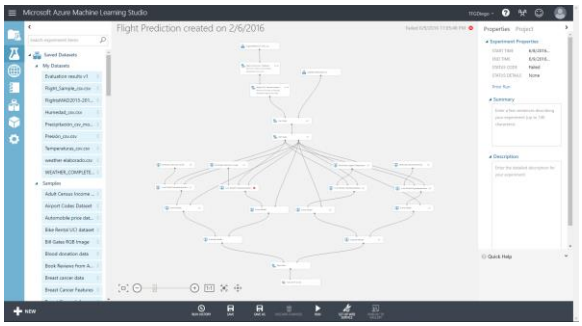


Figure 6. Algorithm selection, Machine Learning Studio.

This second phase allows comparing the accuracy, success rate for each of the three categories and execution time of the model with each algorithm.

Model deployment: after choosing the most accurate algorithm, the next phase is creating the predictive model itself. This step includes setting up a training phase for the model using the data available. Finally, the model is deployed implementing a 'Web Service', which allows accessing the model as a web resource similar to an applet. As a last step, an Excel book is configured as the input of the query and output of the prediction results in order to access the data and prediction results in a more familiar and simple environment.

3. RESULTS

3.1. Algorithm selection

After running the algorithm selection model, the results defining the most accurate one are the following:

Multiclass decision jungle: the part of the model evaluating this algorithm did not execute correctly, so it is not considered as an option to apply in the model.

Multiclass logistic regression: results show 85,65% average precision, but success rate when classifying delayed and cancelled flights is low.

Multiclass neural network: results show 85,66% average precision, but success rate when classifying delayed and cancelled flights is still low.

Multiclass decision forest: results show 85,82% average precision. This is the most accurate algorithm when predicting delayed and cancelled flights.

The chosen algorithm was the last one, as it is the most successful in terms of predicting delay and cancellation of the flights. The specific results are analyzed in the next section of the summary.

3.2. Prediction results

Results obtained after the selection model execution using 'Multiclass decision forest' are presented below.

First, Machine Learning Studio evaluates the model with a tool that provides different parameters measuring the model's performance. Figure 7 contains these parameters.

Overall accuracy	0.787319
Average accuracy	0.858213
Micro-averaged precision	0.787319
Macro-averaged precision	0.797411
Micro-averaged recall	0.787319
Macro-averaged recall	0.365375

Figure 7. Evaluation metrics for the predictive model.

Besides, Machine Learning provides a more visual way of evaluation called confusion matrix. With this method it is easy to understand which is the answer of the model to each of the three possible cases ('On time', 'Delayed' or 'Cancelled'). The confusion matrix for the model is presented below, in figure 8.

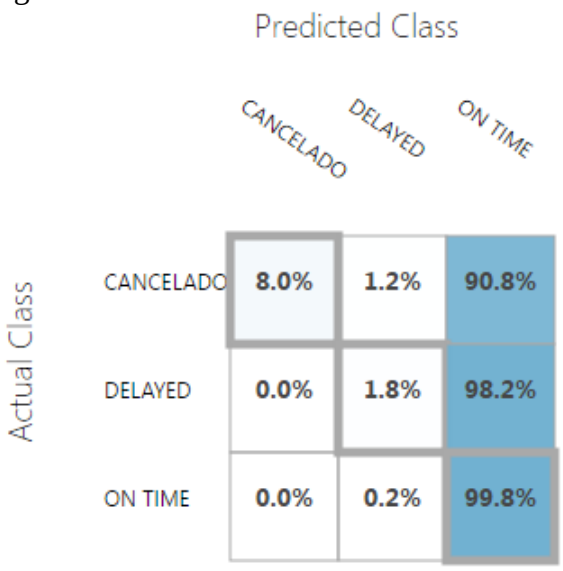


Figure 8. Confusion matrix for the predictive model.

There is further elaboration on conclusions obtained from the results in the next section of the summary.

4. CONCLUSIONS

To end with, as a conclusion, the tasks and goals achieved during the development of the project are analyzed, and the results obtained are explained and summarized. Also, possible improvements to implement in the future are suggested.

During the project, the following tasks and goals have been achieved:

- Real-time weather data capturing with the Sparkfun board has been achieved successfully. This was the first goal defined at the start of the project. The frequency and accuracy of the data capturing has been excellent.
- Correct configuration of the cloud environment and connection of the device with the environment have been successfully achieved. Device-to-cloud data transmission has been successful as well, and data consolidation has been implemented in the cloud as an added value.
- Creation of a predictive model based on the cloud, implementing all the usual phases of these projects (data processing, algorithm selection and deployment of the model) has been successfully achieved. The great complexity of the model has been present throughout all the project. Some possible improvements have been suggested in order to increase precision of the prediction results in the future.
- Creating an 'end-to-end' project covering with enough detail a great variety of technologies and resources has been successfully achieved. These include all the steps of the project, from the initial data capturing to the prediction results analysis.

The conclusions obtained from the results analysis are presented next:

- The great complexity of this type of model stands out, not only because of the huge amount of options available during configuration, but also because of the need of constant troubleshooting, error correction and execution times of the model.
- After analyzing the results, the conclusion is weather conditions are only one of many factors affecting flight delay and cancellation, and this is the main cause of the success rate in these cases being lower.

Last, the implementation of the following solutions could be considered in the future:

- To improve the accuracy of the prediction, it would be necessary to include in the model data related to other delay and cancellation causes. Some of these may be management issues, passenger delays, other flights delays, runway availability, technical failures in the planes, bureaucratic and political reasons and many others. The main advantage this model has is the possibility to easy implement new data, after a simple pretreatment and formatting phase.
- Another option to consider in the future to improve user experience (who is the final receiver of the prediction), would be creating a mobile app implementing the model and presenting the prediction results in an easy to understand way.

5. REFERENCES

[1] https://en.wikipedia.org/wiki/Sensor

[2] https://azure.microsoft.com/en-us/overview/what-is-azure/

[3] https://en.wikipedia.org/wiki/Internet_of_things

[4] https://azure.microsoft.com/en-us/services/iot-hub/

[5] https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-choice/

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MODELO PREDICTIVO. MACHINE LEARNING APLICADO AL ANÁLISIS DE DATOS CLIMÁTICOS CAPTURADOS POR UNA PLACA SPARKFUN.

PROYECTO FIN DE GRADO

Autor: Diego Iribarren Baró

Director: Jaime Pereña Pinedo

Índice General

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MODELO PREDICTIVO. MACHINE LEARNING APLICADO AL ANÁLISIS DE DATOS CLIMÁTICOS CAPTURADOS POR UNA PLACA SPARKFUN.

Documento 1

MEMORIA

Autor: Diego Iribarren Baró

Director: Jaime Pereña Pinedo

Índice de la memoria

Índice de figuras

1. Introducción

El auge de las nuevas tecnologías unido al desarrollo y mejora de la Ingeniería, han propiciado la aparición de aplicaciones conjuntas de estas dos disciplinas. Por una parte, el perfeccionamiento de los sensores permite la obtención cada vez más precisa de datos de todo tipo, y su reducido tamaño facilita la integración de dichos sensores en casi cualquier dispositivo. Por otra parte, el desarrollo de la computación en la nube ha supuesto una nueva dimensión en el almacenamiento y análisis de grandes cantidades de datos con unos costes mucho menores y a una velocidad mucho mayor.

El Proyecto que se presenta a continuación, desarrolla una de las posibles aplicaciones conjuntas de la captura de datos con sensores y su almacenamiento y análisis en la nube. El resultado del proyecto es un modelo predictivo que determina, mediante el análisis de un conjunto acotado de datos, si las condiciones climáticas actuales pueden causar el retraso o cancelación de un vuelo determinado.

La característica más destacable del proyecto es su amplitud, lo que lo convierte en un proyecto 'end to end' que cubre todo el proceso, desde la obtención de datos hasta el análisis final del resultado del modelo.

1.1. Estudio de las tecnologías existentes

Para entender mejor el origen y motivación de este proyecto, se analizarán brevemente los dos componentes fundamentales que se han usado en él: los sensores aplicados a la obtención de parámetros físicos y la computación en la nube combinada con la tecnología 'Big Data' y el Internet de las Cosas (en inglés 'Internet of Things' o 'IoT').

1.1.1. Sensores

Definición

Un sensor es un dispositivo que mide una variable física y la convierte en una variable eléctrica. Están siempre en contacto con la variable que miden, consiguiendo una frecuencia de obtención de medidas muy alta [1].

Origen

Hay cierta incertidumbre en cuanto al origen y primer creador de los sensores. El primer sensor de proximidad data de 1958, y aún hoy sigue usándose.

Figura 1. Sensor de luz. Fuente: prometec.net

La creación del primer sensor inteligente se atribuye a la empresa Honeywell, en 1969. Ese mismo año, Willard Boyle inventó junto con George Smith el sensor CCD, que también se usa actualmente en las cámaras de fotos [2].

Desarrollo

A lo largo de los años, la industria ha creado y desarrollado sensores para casi cualquier magnitud que queramos medir: temperatura, luminosidad, distancia, aceleración, inclinación, desplazamiento, presión, torsión, humedad, movimiento… El principal objetivo es construir sensores más precisos y, según su campo de aplicación, más pequeños. Empresas como Honeywell, Microsens o ZMD son algunas de las más destacadas en este apartado a nivel mundial.

Actualidad

A día de hoy, los sensores juegan un papel fundamental en nuestra vida cotidiana. El desarrollo continuo de las tecnologías de la información y la comunicación (TIC) es una de las principales causas, ya que ha supuesto una demanda constante de mejora y perfeccionamiento de los sensores que contienen todos los dispositivos que usamos día a día. El deseo continuo de mejorar en todos los aspectos (seguridad, transporte, hogar, privacidad, medicina) y el proceso de digitalización en el que la sociedad entera se encuentra inmersa han sido otros incentivos para el desarrollo de sensores.

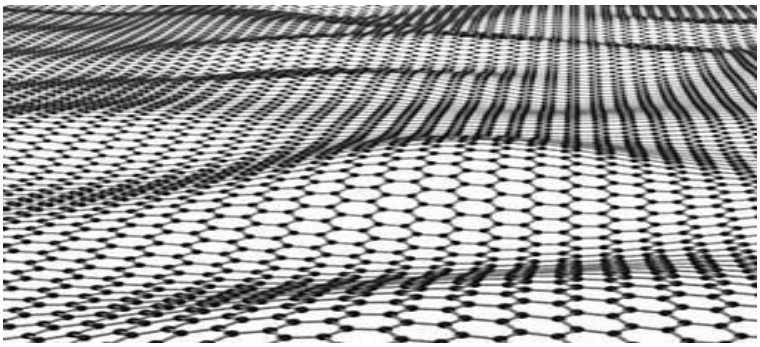


Figura 2. Estructura de grafeno. Fuente: grafeno.com

En la actualidad, el desarrollo de la nanotecnología ha tenido también un gran impacto, ya que el uso del grafeno permite cumplir dos objetivos clave como son abaratar costes y disminuir el tamaño de los sensores. Aunque su descubrimiento data de 1949, el interés que ha suscitado recientemente indica que este campo tendrá un mayor desarrollo en los próximos años.

Aplicaciones

Podría decirse que los sensores pueden aplicarse a casi cualquier ámbito o industria en los que podamos pensar. En los últimos años, se han desarrollado los sensores ya existentes y se ha trabajado en otros nuevos. Desde sensores de uso agrícola (clima) hasta sensores encargados de determinar ciertos comportamientos de un vehículo en caso de accidente, los campos de utilización de sensores son prácticamente infinitos, siendo los más habituales los ya citados: automoción, seguridad y comunicaciones.

Un ejemplo de aplicación de estas tecnologías es la integración de 'IoT' en forma de 'Machine Learning' de la empresa ThyssenKrupp. Thyssen conectó a la nube miles de sensores y sistemas de sus ascensores que supervisan todo, desde la temperatura del motor hasta la alineación de los ejes, la velocidad de la cabina y el funcionamiento de las puertas. Ahora pueden capturar los datos, transmitirlos a la nube y combinarlos en un único panel de control que ofrece dos grupos de datos: alarmas, que indican un problema inmediato, y eventos, que se almacenan y se usan con fines de administración. La solución aporta a los técnicos una gran capacidad de diagnóstico inmediato y una visualización completa de los datos en tiempo real [3].

En este proyecto

En este proyecto se usarán los sensores incluidos en la placa Weather Shield Sparkfun Redboard [4]:

- Sensor de temperatura (HTU21D).
- Sensor de presión (MPL3115A2).
- Sensor de humedad relativa (HTU21D).
- Sensor de luminosidad (ALS-PT19).

Figura 3. Weather Shield. Fuente: sparkfun.com

Además, la placa cuenta con la opción de añadir sensores de viento y lluvia, cuya posible inclusión se ha descartado debido a la complejidad que esto supondría en cuanto al aumento en el número de medidas y variables a analizar por el modelo. En un futuro podrían añadirse dichos sensores para poder realizar una predicción más precisa, utilizando a su vez un modelo más complejo.

1.1.2. 'Cloud Computing', 'Big Data' y 'IoT'

Definición

La **computación en la nube** (en inglés 'Cloud Computing') es un sistema de prestación de servicios en el que se ofrece un catálogo de funciones a las que cualquier usuario pueden acceder sin tener un alto nivel de conocimientos en la gestión de los recursos que usan.

Se caracteriza por tener una gran flexibilidad y adaptabilidad, y su principal ventaja es el pago por consumo. Esta tecnología resulta muy útil, debido a que aporta una gran elasticidad a las empresas en caso de demandas no previsibles o picos de trabajo. Otra gran ventaja de este sistema es la facilidad de acceso, ya que con el desarrollo actual de las nuevas tecnologías se puede disponer de la información o servicio mediante una

conexión a Internet desde cualquier dispositivo móvil o fijo ubicado en cualquier lugar del mundo [5].

Este modelo virtual tiene una base física: un conjunto de servidores que mantiene la infraestructura en funcionamiento continuo, garantizando un mejor tiempo de actividad y una mayor invulnerabilidad frente a ataques, todo ello a un menor coste. Estos servidores suelen estar agrupados en centros de datos (en inglés 'datacenters') que pueden estar alojados en distintas zonas del planeta, independientemente de los lugares de acceso a los servicios. La gran dimensión de estos centros de datos permite aplicar economías de escala y reusar hardware para abaratar costes y mejorar el rendimiento [6].

'**Big Data'** hace referencia a una cantidad de datos tal que supera la capacidad del software utilizado habitualmente para ser capturados, tratados, administrados y procesados en un tiempo razonable, y a los procedimientos usados para encontrar patrones repetitivos dentro de esos datos. Esta disciplina se enmarca en el sector de las TIC y se ocupa de todas las actividades relacionadas con los sistemas que manipulan grandes conjuntos de datos.

El **Internet de las Cosas** (en inglés 'Internet of Things' o 'IoT') es un concepto que se refiere a la interconexión digital de objetos cotidianos con Internet mediante etiquetas de radio de baja potencia (RFID o 'Radio Frequency IDentification'). De esta forma, podrían ser identificados y gestionados por otros equipos (ordenadores), de la misma manera que si lo fuesen por seres humanos.

Origen

La **nube** tiene su origen en proveedores de servicio de Internet a gran escala (Microsoft, Google, Amazon…) que construyeron sus propias infraestructuras. Todos tenían un modelo común: una arquitectura basada en un sistema de servicios virtuales de IT distribuidos horizontalmente y escalados masivamente.

La tecnología '**Big Data'** está estrechamente relacionada con la nube, ya que el almacenamiento, procesamiento y tratamiento de datos suele realizarse en este tipo de plataforma debido a las ventajas que presenta en cuanto a velocidad, capacidad, facilidad de uso y bajos costes.

El origen del '**IoT'** se sitúa en 1999, cuando Bill Joy habló por primera vez de la comunicación D2D ('Device to Device'). Sin embargo, hasta que Kevin Ashton propuso el término Internet de las Cosas (también en 1999), la industria no le dio una oportunidad real a este concepto.

En un artículo de 2009, Ashton hizo la siguiente declaración:

> *"Los ordenadores actuales (y por tanto Internet) dependen de los seres humanos para recabar información. Una mayoría de los casi 50 petabytes (10^{12} bytes) de datos disponibles en Internet fueron inicialmente creados por humanos a base de teclear, presionar un botón, toar una imagen digital o escanear un código de barras. El problema es que las personas tienen tiempo, atención y precisión limitados, lo que significa que no son muy buenos a la hora de conseguir información sobre cosas en el mundo real. Las ideas y la información son importantes, pero las cosas cotidianas tienen mucho más valor. Si tuviéramos ordenadores que supieran todo lo que tuvieran que saber sobre las "cosas", mediante el uso de datos que ellos mismos pudieran recoger sin nuestra ayuda, nosotros podríamos monitorizar, contar y localizar todo a nuestro alrededor. De esta manera se reducirían increíblemente gastos, pérdidas y costes. Sabríamos cuándo reemplazar, reparar o recuperar lo que fuera, así como conocer si su funcionamiento estuviera siendo correcto."*

Desarrollo

Aunque el desarrollo del '**cloud computing**' comenzó en torno a los años sesenta, hasta los noventa no se dispuso de un ancho de banda significativo, por lo que el desarrollo fue tardío. En 1999, Salesforce introdujo por primera vez el concepto de ejecución de aplicaciones empresariales a través de una página web. A partir de ese momento, tanto especialistas como empresas tradicionales de software comenzaron a publicar sus aplicaciones a través de Internet. Posteriormente, en 2002 Amazon habló de un conjunto de servicios basados en la nube, incluyendo almacenamiento y computación, para en 2006 lanzar su Elastic Compute Cloud (EC2): un servicio de alquiler de equipos que permitía a los usuarios ejecutar sus propias aplicaciones informáticas en ellos. En 2009, Microsoft, Google y otros gigantes de la tecnología empezaron a ofrecer aplicaciones basadas en navegador. Según los expertos, el hecho de que estas compañías ofrecieran servicios de este tipo de manera segura y sencilla para el consumidor fue la principal causa de la aceptación de los servicios online.

"El PC de escritorio está muerto. Bienvenido a la nube de Internet, donde un número enorme de instalaciones en todo el planeta almacenarán todos los datos que usted podrá usar alguna vez en su vida."

George Gilder (2006)

Respecto a este tema, es interesante analizar el cambio que se ha producido en los últimos años en el reparto de los servidores existentes en el mercado. En la figura se puede observar que a partir del año 2011 hay un cambio drástico en la tendencia del mercado, que se acentúa a partir de 2012.

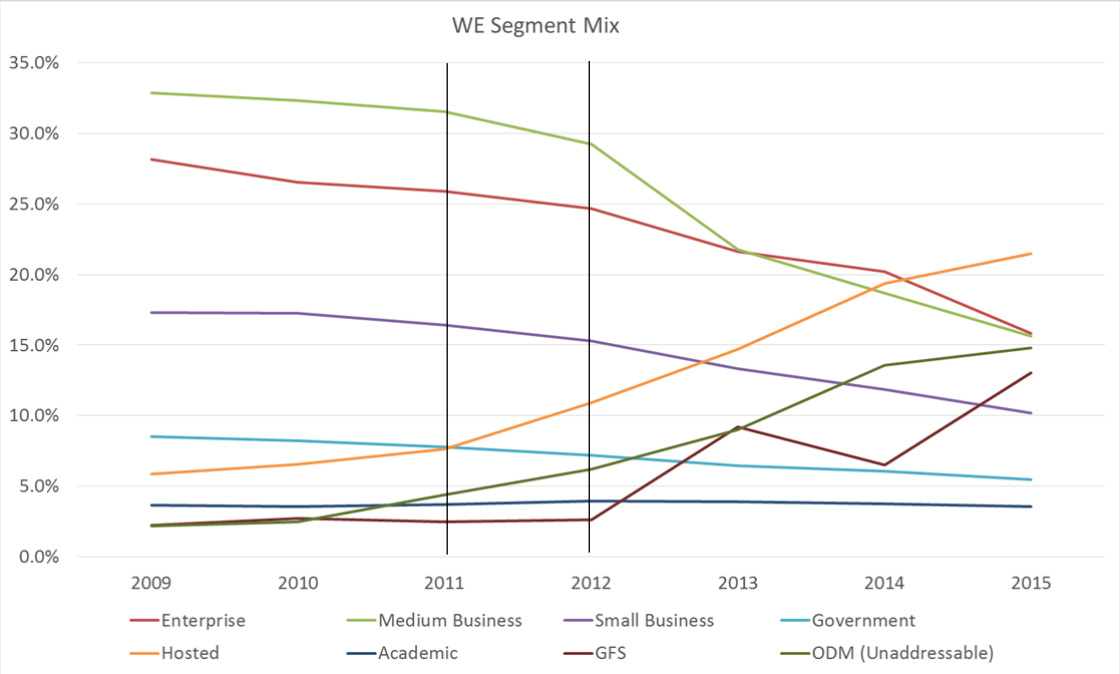


Figura 4: Análisis de la distribución de servidores, Europa del Oeste. Fuente: Microsoft.

Las grandes, medianas y pequeñas empresas cada vez compran menos servidores, así como los organismos públicos. Estos grupos que hace sólo 6 años representaban más del 60% del mercado, han bajado por debajo del 30% en la actualidad. En contraposición a este movimiento, en los entornos en la nube (hosted, ODM, GFS) se ha producido un enorme aumento en el número de servidores, pasando del 15% en 2010 hasta casi el 50% actualmente.

El principal cambio que ofrece la computación en la nube es el aumento del número de servicios basados directamente en la red. Esto genera grandes beneficios tanto para los

proveedores, que pueden ofrecer de forma más rápida y eficiente un mayor número de servicios, como para los usuarios, que tienen la posibilidad de acceder a ellos disfrutando de la transparencia e inmediatez del sistema y haciendo uso de un modelo de pago por consumo. Así mismo, este modelo permite al consumidor ahorrar en costes salariales o en costes en inversión económica, ya que no necesita pagar locales, material especializado, etc.

La infraestructura tecnológica sobre la que se apoya la computación en la nube tiene las siguientes características:

- Posee un alto grado de automatización.
- Permite una rápida movilización de los recursos.
- Tiene una elevada capacidad de adaptación para atender una demanda variable.
- Utiliza un sistema de virtualización avanzada.
- El cálculo del precio es flexible, en función del consumo realizado.
- Evita además el uso fraudulento del software y la piratería.

En cuanto a la tecnología '**Big Data'**, el volumen de datos crece constantemente. En 2012 se estimaba su tamaño en el orden de terabytes (10^{12}) hasta varios petabytes (10^{15}) de datos en un único conjunto de datos. El límite superior de procesamiento también ha ido creciendo a lo largo de los años. De esta forma, los límites fijados en 2008 alcanzaban el orden de zettabytes (10^{21}).

> *Un iPhone hoy en día tiene más capacidad de cómputo que la NASA cuando el hombre llegó a la luna. En 2012 se generaron unos 3×10^{18} bytes de datos al día.*

El campo de estudio más activo de '**IoT'** es el que se basa en la conexión de dispositivos a la red a través de señales de radio de baja potencia. El principal motivo para la utilización de este sistema de conexión es que las señales de este tipo no necesitan ni WiFi ni Bluetooth. Sin embargo, actualmente se están investigando distintas alternativas que requieren una menor cantidad de energía y que resultan más baratas, bajo el nombre de "Chirp Networks".

La empresa estadounidense Cisco está desarrollando en gran medida la iniciativa del 'IoT'. En esta línea, ha creado un contador dinámico de conexiones que le permite estimar en tiempo real el número de "cosas" conectadas a Internet desde Julio de 2013 hasta 2020 [7].

El desarrollo del concepto de 'IoT' podría cambiar el mundo como lo entendemos actualmente.

Figura 5. Internet of Things. Fuente: www.thingscity.com

Si los libros, termostatos, neveras, paquetería, lámparas, botiquines, elementos mecánicos y un largo etcétera estuvieran conectados a Internet y equipados con dispositivos de identificación, no existirían artículos fuera de stock o medicinas caducadas, sabríamos exactamente la ubicación, condiciones de funcionamiento, datos de consumo y compra de productos en todo el mundo y tendríamos acceso a una cantidad inmensa de información sobre las cosas que utilizamos todos los días.

Actualidad

El concepto de **nube informática** es muy amplio, y abarca casi todos los posibles tipos de servicio en línea, pero hoy en día las empresas ofrecen principalmente tres modalidades: Software como Servicio (*'SaaS'*), Plataforma como Servicio (*'PaaS'*) e Infraestructura como Servicio (*'IaaS'*).

'SaaS' es un modelo de distribución de software en el que las aplicaciones están alojadas por una compañía o proveedor de servicio y puestas a disposición de los usuarios a

través de Internet. Es la capa más alta, y consiste en una aplicación completa sobre la que el usuario no tiene control. Esto elimina la necesidad de instalar la aplicación en sus propios servidores y los costes de soporte y mantenimiento de hardware y software.

'PaaS' es un modelo de servicio en el que se ofrece al usuario una plataforma de desarrollo y las herramientas de programación. Es la capa intermedia, y permite desarrollar aplicaciones con plena capacidad de configuración, pero sin control sobre la infraestructura. Puede dar servicio a todas las fases del ciclo de desarrollo y pruebas del software con una gran flexibilidad, aunque las capacidades del proveedor pueden ser restrictivas. Los ejemplos más habituales son Google App Engine y Microsoft Azure (el que se usará en este proyecto).

'IaaS' consiste en la tercerización de los equipos utilizados para apoyar las operaciones, incluidos el almacenamiento, hardware, servidores y componentes de red. Se encuentra en la capa inferior, y es el medio de proporcionar almacenamiento básico y capacidad de computación. En este apartado, cabe destacar Amazon Web Services (AWS), que representa la mayor fuente de beneficios para Amazon. En 2015 generó unos ingresos de 7.88 billones de dólares, y un beneficio de explotación de 1.86 billones de dólares. Estas cifras significaron además un crecimiento anual del 70% y el 182% respectivamente. Para ilustrar de forma clara la ventaja que Amazon tiene frente a sus competidores en materia de IaaS, Gartner afirmó que AWS tiene diez veces la capacidad de sus 14 mayores competidores juntos (incluyendo empresas como Microsoft, IBM o Google).

En una clasificación de otro tipo, la nube puede ser *pública*, *privada* o *híbrida*.

La *nube pública* está mantenida y gestionada por terceros ajenos a la compañía cliente. Datos y procesos de varios clientes pueden estar corriendo a la vez en el mismo servidor, red y sistemas de almacenamiento. El proveedor de servicios es el propietario de toda la infraestructura en sus centros de datos y sólo hay acceso remoto a los servicios.

La *nube privada* está en una infraestructura bajo demanda, gestionada para un solo cliente que es propietario del servidor, red y disco y decide qué usuarios la utilizan. Son

una buena opción para compañías que necesitan alta protección de datos, ya que al administrar internamente estos servicios mantienen la privacidad de su información.

La *nube híbrida* es una combinación de los dos modelos anteriores, en la que el usuario es propietario de unas partes y comparte otras, aunque de una manera controlada. Las nubes híbridas incluyen escalabilidad aprovisionada externamente, pero añaden la complejidad de determinar la distribución de las aplicaciones a través de estos ambientes diferentes.

El uso de '**Big Data'** presenta una serie de dificultades vinculadas a la gestión de estas grandes cantidades de datos que tienen lugar durante la recolección y el almacenamiento, búsqueda, compartición, análisis, y visualización. El uso de este recurso se justifica con la necesidad de incluir dicha información en la creación de informes estadísticos y modelos predictivos utilizados en diversas materias, como los análisis de negocio, análisis publicitarios, los datos de enfermedades infecciosas, el espionaje y seguimiento a la población o la lucha contra el crimen organizado. Hoy en día, los campos de aplicación más habituales de este recurso son el ámbito empresarial (redes sociales, consumo, resultados), el deportivo (a nivel profesional o aficionado) y la investigación (salud, defensa, sostenibilidad…).

Figura 6. Big Data. Fuente: www.silicon.es

Cuando hablamos de 'Big Data' podemos clasificar los datos en tres grandes grupos.

Los *datos estructurados* son aquellos con un formato y tamaño definido, como los números o las fechas. Se almacenan en tablas (bases de datos, hojas de cálculo…).

Los *datos no estructurados* no tienen un formato específico, sino que se encuentran tal y como fueron obtenidos (documentos de texto, documentos multimedia, emails…).

Los *datos semiestructurados* tienen marcadores para separar los elementos, pero no se limitan a un campo determinado (HTML, XML, JSON…).

La etapa de desarrollo tecnológico en la que vivimos ha acelerado el crecimiento de la cantidad de datos que se generan. La capacidad de procesamiento de datos ha aumentado en similar medida, y las previsiones actuales nos adelantan hechos increíbles que serán reales en un futuro más cercano de lo que creemos. Según estas estimaciones, en 2025 se tendrá capacidad de cálculo suficiente como para realizar simulaciones completas del cerebro humano y todas sus neuronas. Hacia 2030, la capacidad de los superordenadores llegará a los zettaflops, lo que significa por ejemplo que las previsiones meteorológicas para un tiempo de dos semanas tendrán una precisión del 99%. El continuo aumento de capacidad haría posible la simulación de millones de cerebros humanos de manera simultánea en 2050.

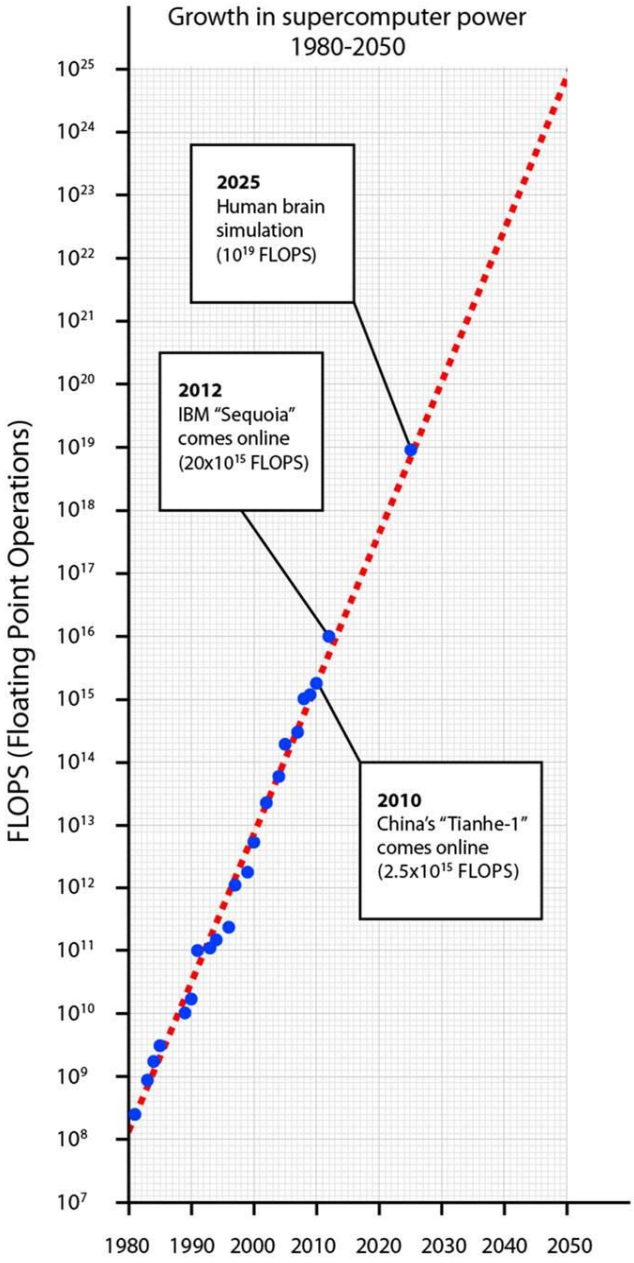


Figura 7. Capacidad de computación de superordenadores. Fuente: futuretimeline.net

Actualmente, el término '**IoT'** se usa para definir una conexión avanzada de dispositivos, sistemas y servicios que va más allá del tradicional M2M (máquina a máquina) y que cubre una amplia variedad de protocolos, dominios y aplicaciones.

Se calcula que todo ser humano está rodeado de por lo menos 1.000 a 5.000 objetos, por lo que el Internet de las Cosas debería codificar de 50 a 100.000 billones de objetos y seguir su movimiento. Con las aplicaciones de Internet basadas en el protocolo IPv6 se podrían identificar todos los objetos, algo que no se podía hacer con IPv4. Este sistema sería capaz de identificar instantáneamente por medio de un código cualquier tipo de objeto.

> *"El Internet de las Cosas tiene el potencial para cambiar el mundo tal y como hizo la revolución digital hace unas décadas. Tal vez incluso más."*
>
> *Kevin Ashton*

Según la empresa Gartner[8], en 2020 habrá en el mundo aproximadamente 26.000 millones de dispositivos con un sistema de adaptación al IoT, mientras que otras empresas predicen que existirán 30.000 millones de dispositivos inalámbricos conectados a Internet. Hay que tener en cuenta que los estudios relacionados con el IoT están todavía en un punto relativamente temprano de desarrollo, y que sólo se está empezando a explotar todo el potencial de esta idea.

Aplicaciones

Generalmente, el esquema que se utiliza en las aplicaciones que incluyen estos tres componentes ('cloud computing', 'Big Data' y 'IoT') es el siguiente:

Se utiliza la nube como base en cualquiera de las tres variantes que se han analizado (Saas, PaaS o IaaS), y se ejecuta una aplicación de IoT que normalmente genera o requiere el uso de tecnología 'Big Data'.

> *"Casi cualquier cosa puede ser utilizada en la nube"*
>
> *Jamie Turner*

En cuanto a ejemplos concretos, un caso conocido en el que la tecnología 'Big Data' juega un papel principal es el de AccuWeather [9], empresa que elabora predicciones meteorológicas gracias al análisis de grandes cantidades de información. También es destacable el ya mencionado caso de ThyssenKrupp, con sus ascensores conectados a la

nube. Como ejemplo de IaaS, destaca el acuerdo entre SAP Hana y Microsoft [10], por el que la plataforma Azure aloja las aplicaciones de SAP [11] dándoles el soporte para ejecutarlas. Finalmente, como ejemplo de SaaS se puede citar a Salesforce [12], empresa que proporciona una serie de servicios totalmente alojados en la nube a sus clientes entre los que destaca su sistema de CRM (Customer Relationship Management).

En este proyecto

En este proyecto, como se ha descrito en el punto anterior, se usará la plataforma Microsoft Azure, nube híbrida que ofrece servicios 'PaaS'. Por un lado, se creará una interfaz de 'IoT' a la que llegarán en tiempo real los datos obtenidos mediante los sensores de la placa. Por otro lado, se usará la capacidad Machine Learning de Azure para crear, entrenar y desplegar un modelo de predicción basado en datos históricos de meteorología y vuelos ('Big Data').

Por último, se usará Power BI como herramienta de visualización y presentación de datos [13].

El principal valor que aporta este proyecto es la integración de todas las etapas de la aplicación de estas tecnologías: obtención, almacenamiento, procesamiento y tratamiento de datos, creación del modelo en la nube, aplicación de dicho modelo, obtención de resultados y análisis de los mismos. Esto lo convierte en un proyecto 'end to end'.

1.2. Motivación del proyecto

Como se ha visto en el apartado anterior, el desarrollo de los sensores y de las plataformas en la nube ha favorecido la aparición de multitud de aplicaciones conjuntas de ambos recursos. Este proyecto consiste en una aplicación práctica de los avances en los dos campos mencionados, utilizando dos de las muchas alternativas posibles: la placa Sparkfun y los servicios en la nube de Microsoft.

Se podría profundizar mucho más en varias de las áreas involucradas, pero se ha preferido enfocar el proyecto con una visión más amplia que además permita utilizar

todo el abanico de tecnologías desde la generación y captación de la información hasta el tratamiento, ordenamiento, consulta y predicción.

1.3. Objetivos del proyecto

1.3.1. Captación de datos mediante sensores

Este primer objetivo es clave para la correcta ejecución del proyecto, ya que los datos obtenidos son la base a analizar para determinar el resultado del modelo predictivo. Si no se captan correctamente las condiciones climáticas, no podrán compararse con los datos almacenados y el resultado de la predicción podría ser erróneo.

En el recuadro se pueden apreciar las variables que componen un 'log', o registro de todos los datos disponibles.

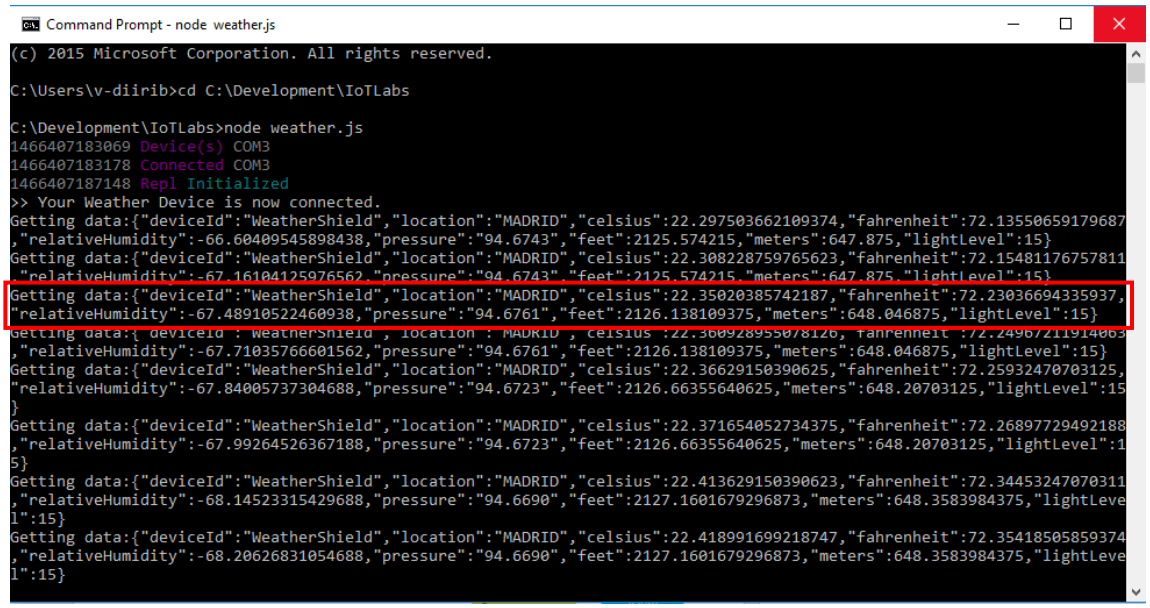


Figura 8. Registro de la toma de datos mediante la placa Sparkfun usando la consola de Windows 10.

Para asegurar que no haya errores en este apartado, se han realizado comprobaciones de los datos medidos en diferentes ambientes.

1.3.2. Transmisión, procesamiento y almacenamiento de datos

Este objetivo tiene una importancia vital en la mayoría de proyectos de este tipo. Sin embargo, en este caso concreto no se ha profundizado mucho en él, debido a que otros objetivos tienen mayor relevancia.

Sí que se han realizado comprobaciones para verificar que la transmisión, procesamiento y almacenamiento de los datos en la nube son correctas, y asegurar que el modelo está recibiendo los datos necesarios para funcionar correctamente.

En el recuadro se puede observar cómo se realiza la importación de los datos medidos por la placa. La consulta creada proporciona la media de todos los parámetros: temperatura (en Celsius y Farenheit), humedad relativa, presión y luminosidad, e incluye además un parámetro de altitud que viene fijado por defecto.

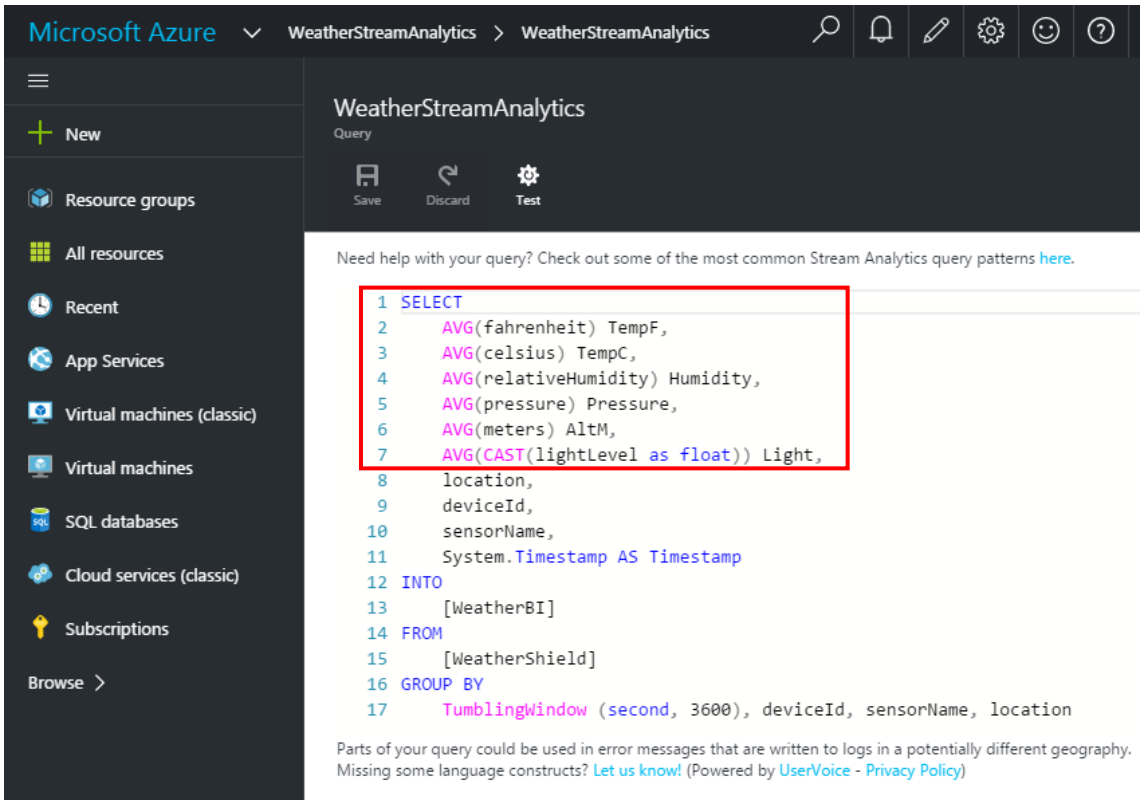


Figura 9. Stream Analytics, entorno de Microsoft Azure para la transmisión de datos a la nube y otros recursos.

1.3.3. Creación de un modelo predictivo en la nube

El modelo predictivo es el núcleo central del proyecto, ya que es el encargado de analizar los datos obtenidos, ejecutar el algoritmo para analizarlos y proporcionar el resultado de la predicción.

Si hubiera fallos en este apartado, el resultado del proyecto sería incorrecto. Para mejorar la solidez del modelo, se han incluido varios bloques de comprobación y se ha entrenado previamente el modelo con los datos de que se dispone. Para ello se han utilizado las herramientas incluidas de forma predeterminada en Azure, que permiten evaluar el modelo analizando parámetros como la precisión total, precisión media y porcentaje de acierto en la clasificación.

En la imagen se presenta una parte de un experimento creado para probar la eficacia de uno de los algoritmos elegidos.

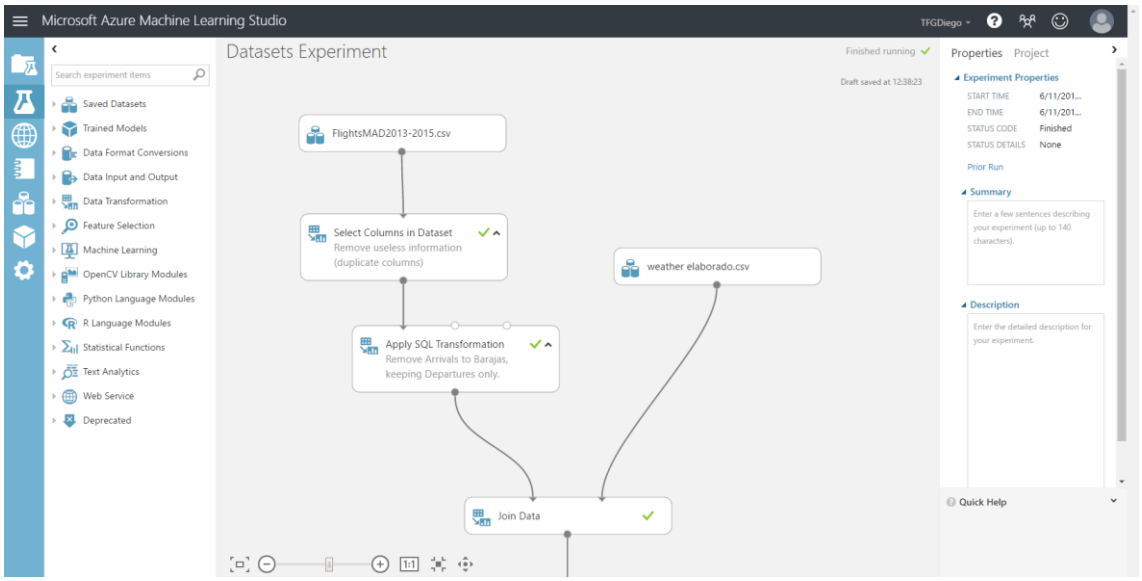


Figura 10. Machine Learning Studio, entorno de Machine Learning de Microsoft Azure.

El modelo predictivo se estudia en detalle en el apartado 5.6. de la memoria.

1.3.4. Presentación visual de los datos

El objetivo último del modelo es determinar si un determinado vuelo va a sufrir cancelaciones o retrasos en unas determinadas condiciones atmosféricas. Para facilitar la interpretación de los datos se ha utilizado Power BI, una herramienta de tratamiento, análisis y presentación de datos. Mediante diferentes gráficas se han elaborado una

serie de paneles de control (en inglés 'dashboard') que hacen más sencilla y visual la interpretación de los datos y resultados del modelo.

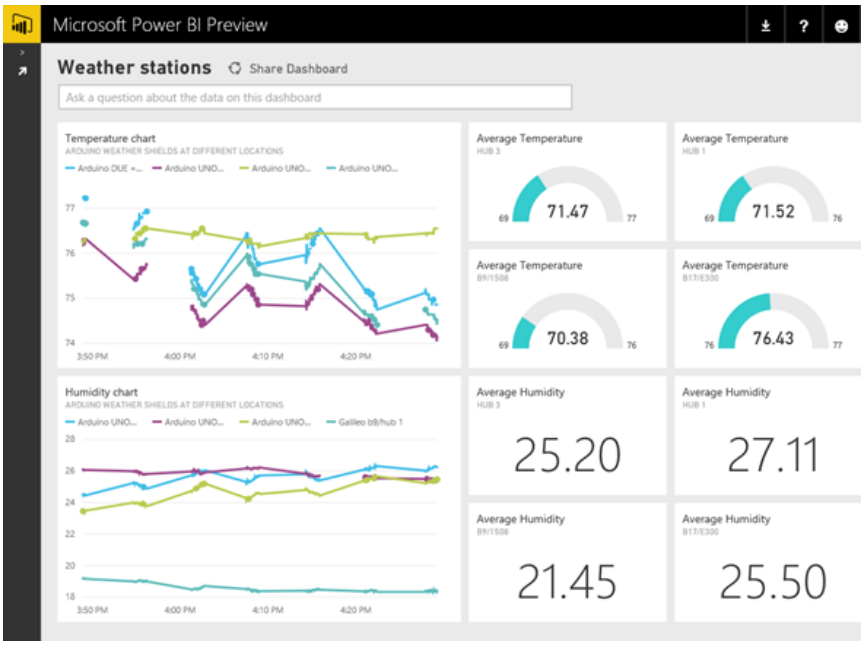


Figura 11. Ejemplo de Dashboard usando PowerBI. Fuente: powerbi.microsoft.com

Por último, se ha decidido utilizar una hoja Excel como entrada de consulta para trabajar en un entorno más familiar y universal.

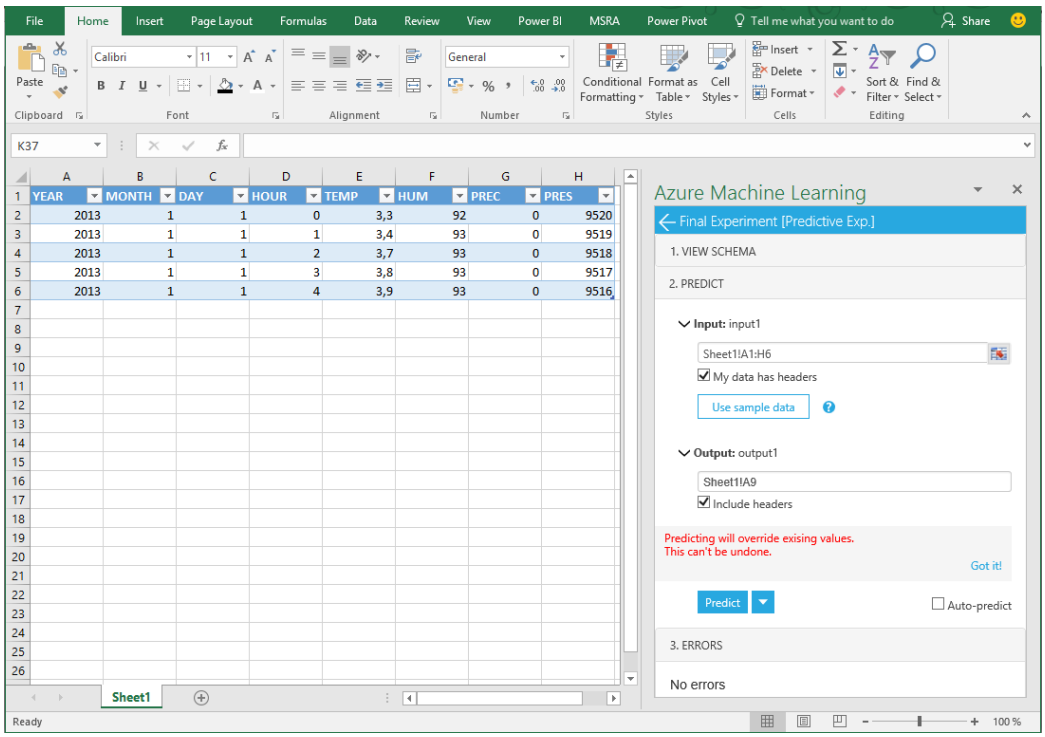


Figura 12. Web service implementado en Excel.

En un siguiente paso podría plantearse crear una aplicación que implementase todos los pasos del proyecto, y cuya interacción con el usuario se limitase a pedir los datos del vuelo y proporcionar el resultado de la predicción, realizando de forma automática la obtención de datos meteorológicos y la ejecución del modelo.

1.4. Metodología de trabajo

A continuación, se presenta de forma breve la metodología que se ha utilizado durante el desarrollo del proyecto, destacando los hitos que marcan la consecución de cada etapa. La descripción detallada de las fases del desarrollo se encuentra en el apartado 2 de esta memoria.

Estudio previo

Esta etapa incluye la búsqueda de información acerca del tema en el que se basa proyecto para poder entender mejor los avances ya realizados y los posibles enfoques del proyecto. Se consideró finalizada cuando se definió el tema del proyecto, aunque la búsqueda de información se prolongó hasta el fin de la redacción de la memoria.

Configuración de la placa

Esta fase incluye tanto el aprendizaje necesario para configurar correctamente la placa como la propia configuración. Se consideró finalizada cuando la placa fue capaz de captar información meteorológica en tiempo real y transmitirla al ordenador correctamente.

Configuración de la nube

Esta fase incluye la familiarización con las herramientas disponibles, la elección de las que se usaron en el proyecto y la configuración de las mismas. Se consideró finalizada cuando se recibieron los datos enviados por la placa correctamente.

Modelo predictivo

Esta fase incluye el estudio de los elementos básicos de un modelo predictivo, los distintos tipos existentes, su funcionamiento, entrenamiento y evaluación, y, por supuesto, su creación y configuración. Se consideró finalizada cuando el modelo fue

capaz de predecir correctamente el estado de los vuelos analizados, y se realizó el despliegue del modelo.

Cronograma y diagrama de Gantt

El desarrollo del proyecto comenzó en Marzo (estancia Erasmus+ en Bruselas durante el primer semestre).

La duración de prácticamente todas las tareas fue la esperada. El único contratiempo surgió a la hora de obtener el histórico de datos de AENA y AEMET, ya que, debido a los procesos internos y a los retrasos habituales a la hora de gestionar este tipo de peticiones, se excedió el tiempo previsto para la consecución de dicha tarea. Este es el principal motivo por el que el proyecto se presenta en segunda convocatoria.

A continuación, se incluyen dos figuras que ilustran el desarrollo del proyecto a lo largo del tiempo. En primer lugar, un cronograma realizado con el programa Microsoft Project que incluye las tareas explicadas previamente y situadas en el tiempo en función de sus fechas de inicio y final.

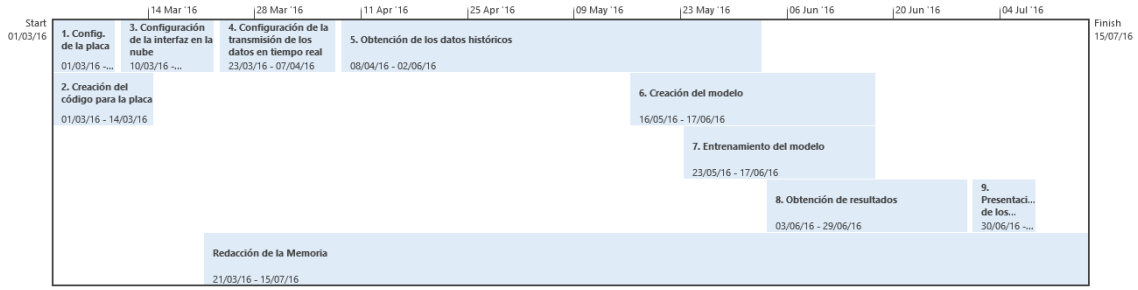


Figura 13. Cronograma del proyecto, realizado en Microsoft Project.

En segundo lugar, se incluye un diagrama de Gantt, que ilustra además las dependencias entre las tareas y la duración de las mismas. Como ya se ha mencionado, se destaca en amarillo la tarea cuya duración excedió el tiempo previsto.

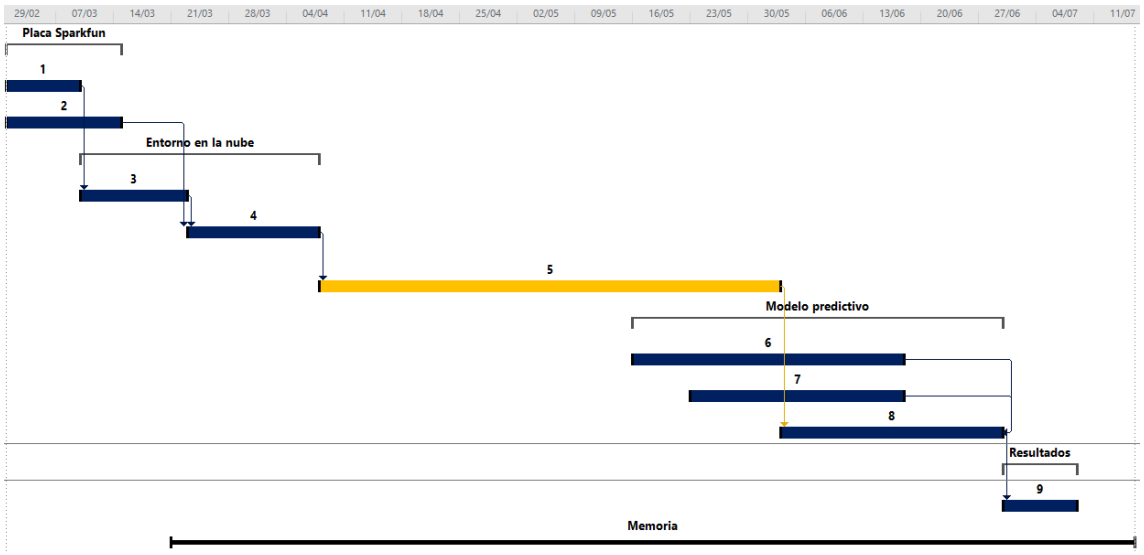


Figura 14. Diagrama de Gantt del proyecto, realizado en Microsoft Project.

1.5. Recursos utilizados

En el desarrollo de este proyecto se ha utilizado una serie de recursos que se pueden clasificar en hardware, software y datos.

1.5.1. Hardware

SparkFun RedBoard: esta placa puede programarse con el IDE Arduino conectándola al ordenador mediante un cable mini-USB. Tiene 14 pines digital I/O y 6 pines PWM, además de 6 pines de entrada analógica. Es compatible con otros accesorios, como el Weather Shield.

Figura 15. Sparkfun RedBoard. Fuente: www.sparkfun.com

SparkFun Weather Shield: este accesorio puede medir la temperatura, humedad relativa, luminosidad y presión gracias a sus sensores de humedad, presión y luz. Además, soporta la conexión de sensores de lluvia y viento e incluso de GPS.

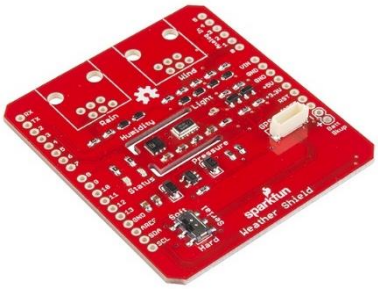


Figura 16. SparkFun Weather Shield. Fuente: www.sparkfun.com

1.5.2. Software

Azure: la plataforma en la nube de Microsoft, Azure, incluye infinidad de recursos al alcance del usuario. En este proyecto se han utilizado los siguientes:

Figura 17. Plataforma Microsoft Azure. Fuente: Microsoft.

IoT Hub: este servicio permite establecer una comunicación bidireccional entre los dispositivos 'IoT' y la nube de forma segura y compatible con diversas plataformas [14].

Stream Analytics: esta herramienta permite el procesamiento de transmisiones en tiempo real en la nube, incluyendo también opciones de análisis y tratamiento de los datos transmitidos [15].

Machine Learning Studio: esta herramienta permite al usuario crear e implementar soluciones en los datos de que dispone [16]. Presenta una interfaz fácilmente configurable y con una gran usabilidad que implementa todas las ventajas de la nube.

PowerBI: este conjunto de aplicaciones de análisis de datos permite realizar informes interactivos de forma sencilla para interpretar la información obtenida [17].

Figura 18. PowerBI, herramienta de tratamiento y presentación de datos de Microsoft.

1.5.3. Datos

Datos históricos de meteorología: se han obtenido datos meteorológicos de los últimos tres años (2013 a 2015) incluyendo temperatura, precipitaciones, presión y humedad relativa de cada hora. "Información elaborada utilizando, entre otras, la suministrada por la Agencia Estatal de Meteorología. Ministerio de Agricultura, Alimentación y Medio Ambiente".

Figura 19. Logo de la AEMET, Agencia Estatal de Meteorología.

Datos históricos de vuelos: se han obtenido datos de todos los vuelos con origen en el Aeropuerto Adolfo Suárez Madrid Barajas en los últimos tres años (2013 a 2015). "Información suministrada por AENA, Aeropuertos Españoles y Navegación Aérea".

Figura 20. Logo de AENA, Aeropuertos Españoles y Navegación Aérea.

2. Desarrollo del proyecto

2.1. Placa Sparkfun

2.1.1. Configuración inicial

Para configurar la placa se han realizado dos operaciones usando el programa Arduino. En primer lugar, se instalaron el IDE de Arduino y los drivers de la conexión USB, y posteriormente se actualizó la Firmata [18].

Figura 21. Actualización de la Firmata de la placa usando Arduino.

2.1.2. Creación del código

En segundo lugar, se usó el programa Visual Studio para escribir el código para la toma de datos. Se crearon dos archivos:

- package.json: permite usar NPM para descargar los módulos Node necesarios.

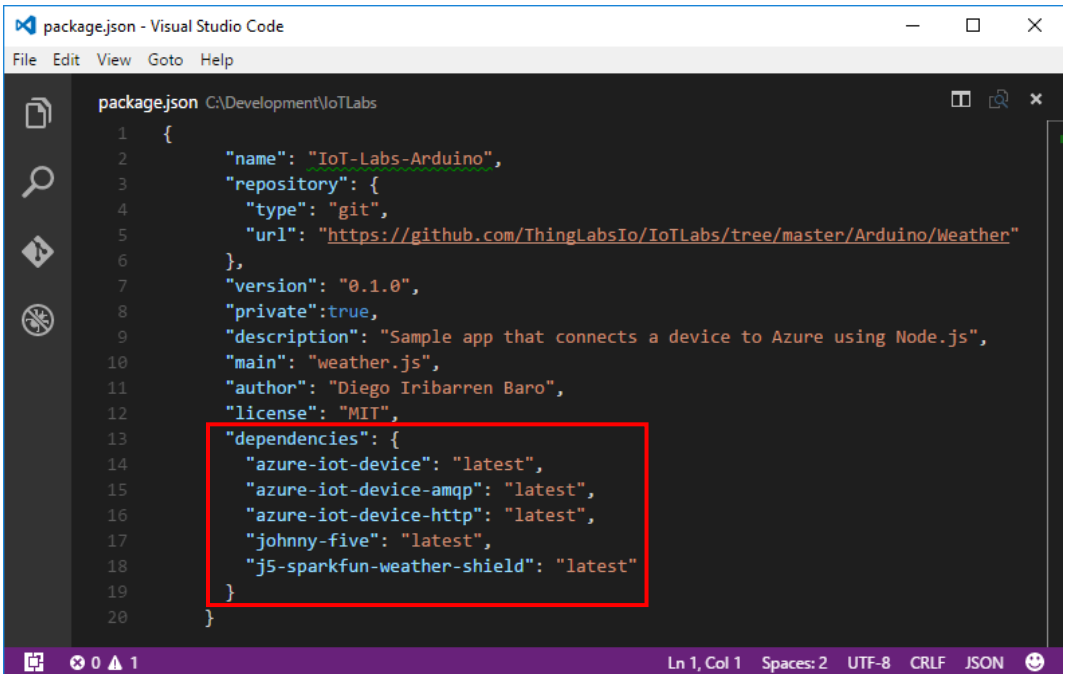


Figura 22. Archivo package.json, escrito y compilado usando Visual Studio.

En el recuadro se puede observar la configuración de todas las dependencias para que se actualicen automáticamente y usen siempre la última versión disponible de software.

- weather.js: contiene el código de la placa en sí, un JavaScript que determina la frecuencia a la que se miden los parámetros meteorológicos, las variables en las que se almacenan estos parámetros, y la configuración necesaria para enviar la información a la nube. Además. en este archivo se define el formato que tendrán los 'logs' de información que se recibirán desde la placa, y el contenido de los mismos, permitiendo modificar todos los parámetros anteriores.

  Como se indica en la parte de código recuadrado, la frecuencia de toma de datos se ha fijado en 1000 milisegundos, ya que esa gran cantidad de datos se consolidará posteriormente. También se ha determinado la altitud a la que se encuentra el dispositivo, aunque la variación de este parámetro tiene un impacto casi inapreciable en este proyecto.

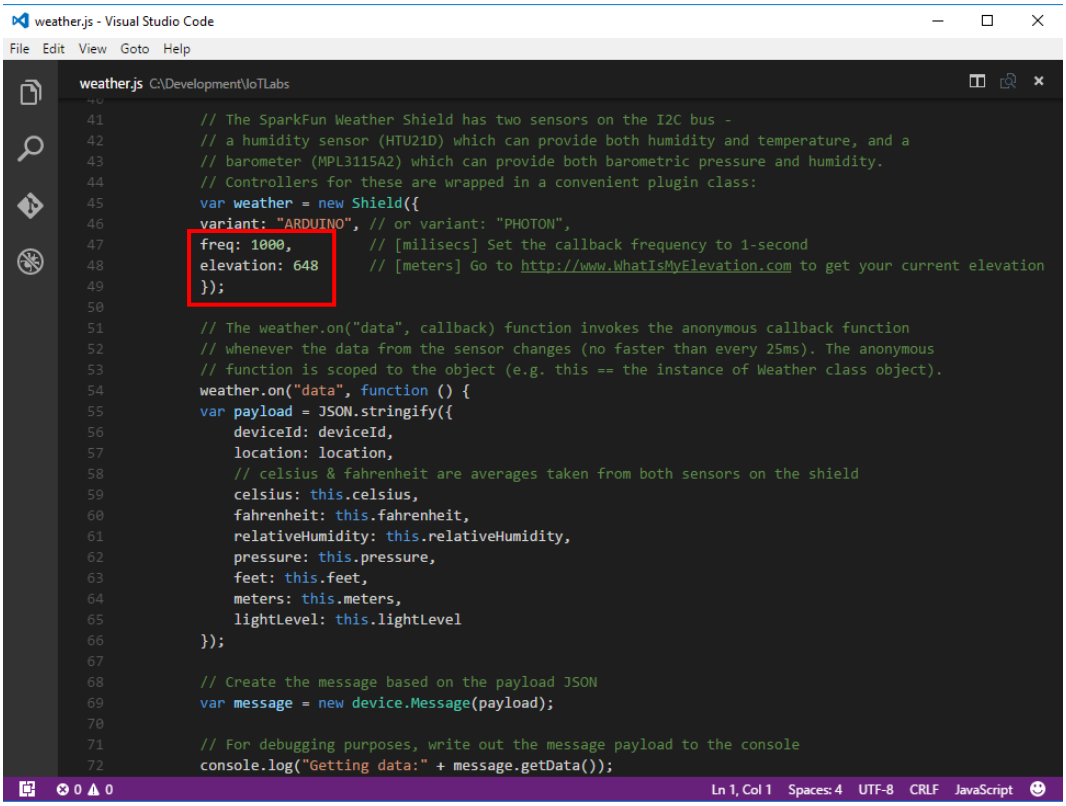


Figura 23. Archivo weather.js, escrito y compilado usando Visual Studio.

2.2. Entorno en la nube

2.2.1. Configuración inicial

Para configurar el entorno en la nube, se creó y configuró un Azure IoT Hub en el portal Azure [19]. Este servicio totalmente gestionado por Azure permite la conexión y comunicación segura y bidireccional entre los dispositivos de IoT que se quieran utilizar y la solución de Azure. Las principales ventajas que presenta son:

- Comunicación dispositivo-nube y nube-dispositivo fiable y escalable.
- Comunicación segura usando credenciales y control de acceso para cada dispositivo.
- Monitorización de la conectividad de dispositivos y gestión de eventos de identificación.
- Inclusión de librerías predeterminadas para los principales lenguajes y plataformas.

En la figura se encuentran todos los recursos de IoT disponibles en la plataforma de Azure, de entre los cuales se ha decidido utilizar el IoT Hub debido a las ventajas que presenta frente a las otras opciones.

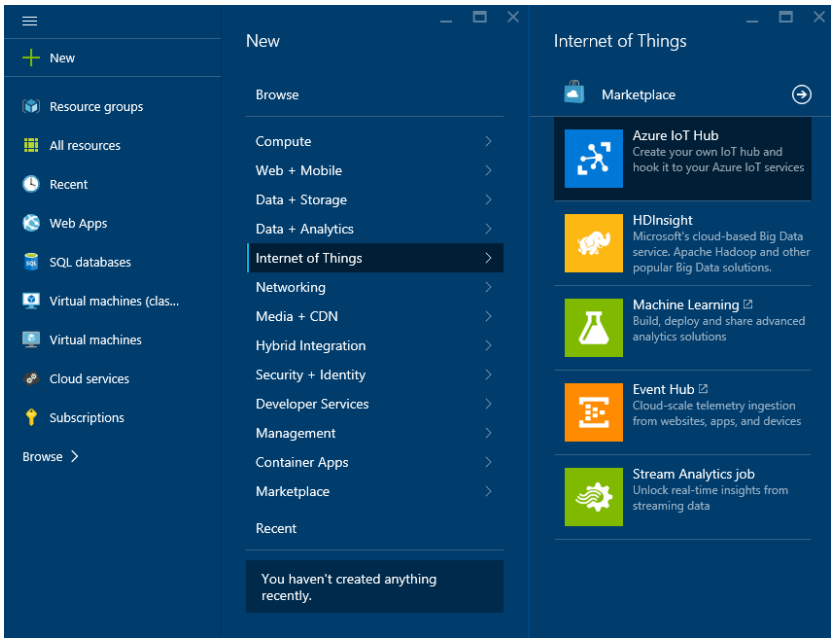


Figura 24. Configuración de Azure IoT Hub.

Una vez configurado el entorno, se conectó el Weather Shield al IoT Hub usando el Azure IoT Hub Device Explorer. Esta herramienta se utiliza para gestionar los distintos dispositivos conectados al Azure IoT Hub. En este caso, al haber un único dispositivo conectado a la nube, su inclusión no es excesivamente relevante, pero en proyectos de mayor escala su utilidad es innegable.

Device Explorer se ejecuta localmente desde un ordenador y se conecta al IoT Hub. Una vez conectado al Hub mediante las claves de conexión correspondientes, se pueden añadir múltiples dispositivos que quedan de igual manera conectados a la nube. Además de la conexión, la herramienta permite la monitorización de comunicaciones 'device-to-cloud' y el envío de mensajes 'cloud-to-device'.

En la imagen se han ocultado las claves de conexión, pero se pueden apreciar el resto de detalles de la configuración de la conexión, como el nombre del IoT Hub al que se conecta. En la pestaña 'Management' se encuentran los dispositivos asociados al Device

Explorer, que como se ha explicado previamente tendrá más relevancia cuanto mayor sea el número de dispositivos utilizados en el proyecto.

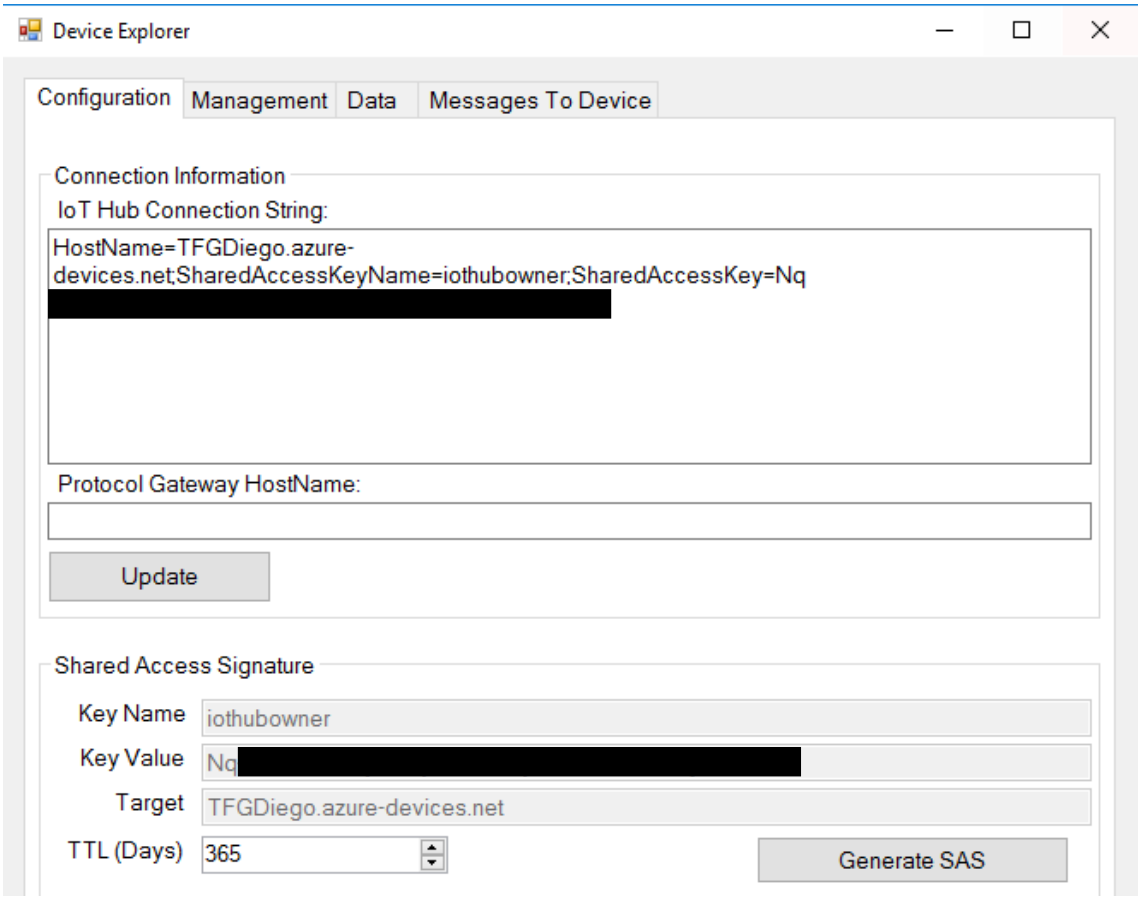


Figura 25. Configuración del dispositivo usando Device Explorer.

De esta forma, la información capturada por la placa Sparkfun y el Weather Shield va a parar a la nube, donde se almacena.

2.2.2. Transmisión de datos

A continuación, se configuró la transmisión de los datos desde la placa hasta PowerBI, la herramienta de procesamiento y visualización de datos de Microsoft. Para ello, se usó la herramienta Stream Analytics, que permite procesar y transmitir datos en tiempo real. Este motor de procesamiento de eventos permite importar datos de varias fuentes distintas, tratarlos de forma sencilla con un lenguaje similar a SQL y monitorizar las comunicaciones entrantes, ajustando la velocidad y escala de procesamiento desde unos pocos kilobytes hasta un gigabyte de eventos por segundo.

En la figura se puede apreciar la facilidad con la que se puede crear este entorno inicialmente, con una inmensa cantidad de opciones a configurar posteriormente.

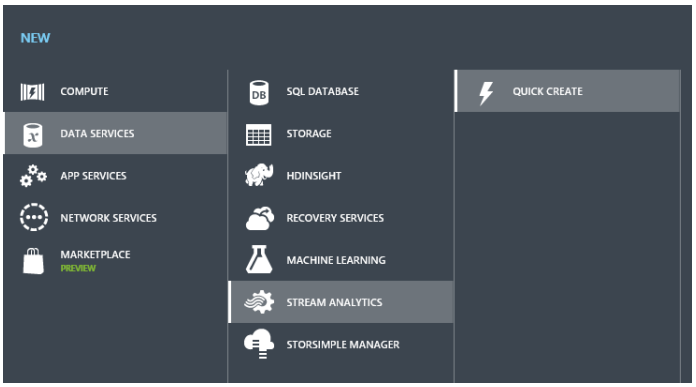


Figura 26. Configuración del entorno de Stream Analytics.

Entre las distintas opciones que ofrece Stream Analytics, se eligió usar como 'input' el dispositivo Weather Shield y como 'output' la herramienta PowerBI, como ya se ha mencionado.

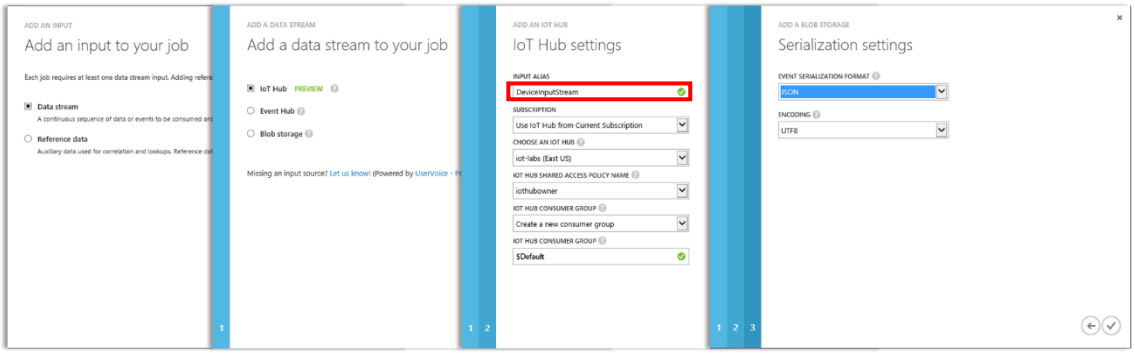


Figura 27. Configuración de la entrada de Stream Analytics.

Figura 28. Configuración de la salida de Stream Analytics.

Por último, se usó la funcionalidad integrada en Stream Analytics que permite escribir una consulta ('query') para importar los datos medidos por la placa, y se configuraron de nuevo los nombres y formatos de alguna de las variables.

En el recuadro se encuentra el parámetro utilizado para consolidar los datos recibidos cada hora (3600 segundos), de forma que se visualice la media horaria y no los datos enviados por la placa cada segundo.

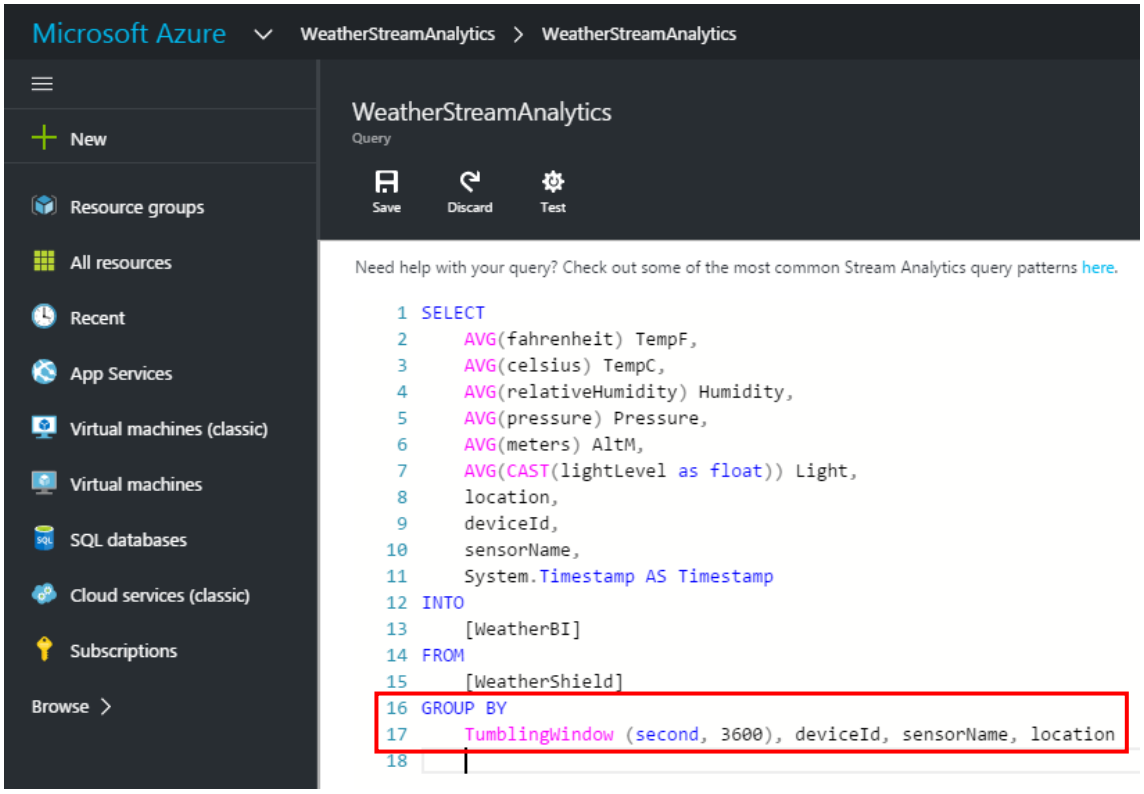


Figura 29. Consulta (query) de Stream Analytics.

2.3. Modelo predictivo

2.3.1. Creación del modelo

El esquema habitual de este tipo de modelos tiene tres fases: preparación de los datos, elección del algoritmo y despliegue del modelo. A continuación, se explica en detalle cada una de las fases, concretando los pasos seguidos en este proyecto.

Preparación de los datos

Para acotar el alcance del proyecto, se decidió restringir ambos conjuntos de datos (vuelos y condiciones meteorológicas) a la zona del aeropuerto Adolfo Suárez Madrid Barajas, e incluir únicamente los últimos 3 años (2013 a 2015).

En primer lugar, se realizó un procesado inicial de los datos brutos para obtener lo que se conoce como datos preparados. Este procesado inicial tiene como objetivo adaptar el formato a las necesidades del modelo y simplificar así su construcción. Para ello se usaron las propias herramientas de Excel (fórmulas, macros y filtros). En el caso de los vuelos, se cambió el formato de las fechas y las horas de salida, se añadió un campo con el estado del vuelo (Cancelado, Retrasado, En hora) y otro con el tiempo de retraso en minutos. Además, se guardó el archivo en formato .csv para poder utilizarlo en la plataforma de Azure. En el caso de los datos meteorológicos, se combinaron los archivos con los distintos registros (temperatura, presión, humedad y precipitaciones) en un único archivo con toda la información clasificada por horas. Por último, se cambió el archivo a formato .csv para poder utilizarlo en el modelo.

Posteriormente, se creó un experimento (nombre que recibe cada archivo en Azure Machine Learning Studio) dedicado a modificar los archivos con el formato adecuado para obtener un conjunto de datos óptimo para alimentar el modelo final. Durante esta fase se usaron bloques como selección de columnas, operaciones matemáticas, edición de metadatos y combinación de conjuntos de datos, que permitieron combinar los archivos inicialmente separados de condiciones meteorológicas y vuelos en un único archivo. Este archivo de datos o 'dataset' es el que se utilizó finalmente como primera entrada para el modelo.

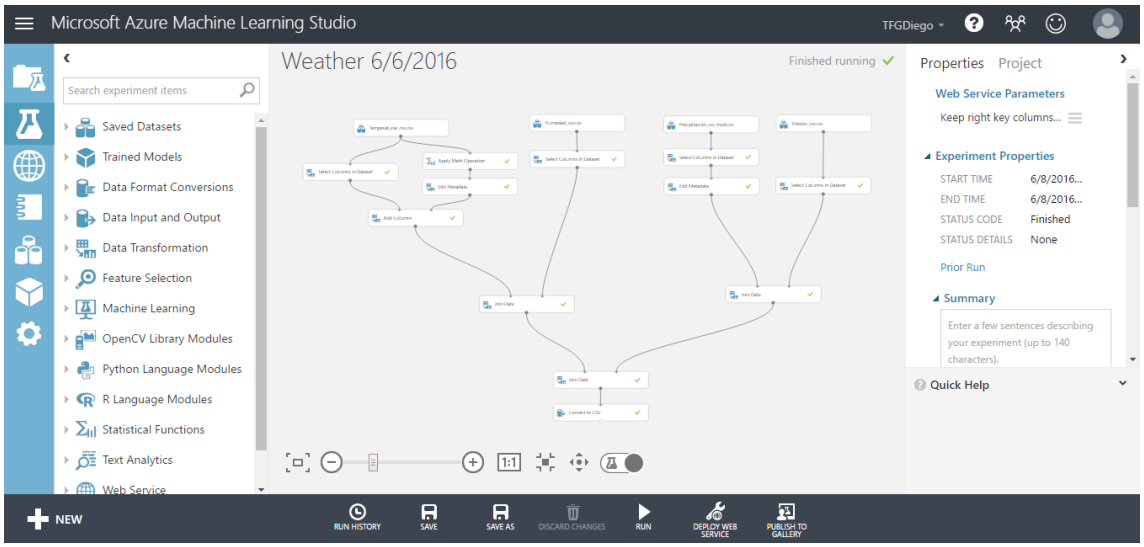


Figura 30. Experimento para la consolidación de los datos.

Elección del algoritmo

La segunda fase consistió en la elección del algoritmo predictivo a utilizar. Con el objetivo de crear un modelo preciso, sólido y sencillo, se siguieron las sugerencias propuestas en la página web de Azure [20] relativas a la elección del algoritmo más adecuado. Dependiendo de la naturaleza del proyecto se pueden elegir distintos algoritmos de entre los disponibles de forma predeterminada en Azure Machine Learning. Este proceso puede ser iterativo también, dado que el algoritmo elegido determinará en gran medida el rendimiento y porcentaje de acierto del modelo.

Para este proyecto se decidió emplear algoritmos de clasificación multiclase. Este tipo de algoritmos, como su propio nombre indica, clasifica cada uno de los casos a analizar en una serie de categorías predeterminadas por el usuario. En este proyecto esas categorías son tres, por lo que la clasificación es multiclase (el estado de los vuelos puede ser 'En hora', 'Retrasado' o 'Cancelado').

Con el propósito de elegir el mejor algoritmo, se creó un nuevo experimento que contenía los cuatro algoritmos más destacados de este tipo: Multiclass Decision Forest, Multiclass Decision Jungle, Multiclass Logistic Regression y Multiclass Neural Network. La plataforma de Machine Learning Studio de Azure cuenta con una serie de bloques (Score Model, Train Model, Evaluate Model, Tune Model Hyperparameters) cuya

función es optimizar los parámetros de cada algoritmo y comparar sus resultados para poder utilizar el más efectivo.

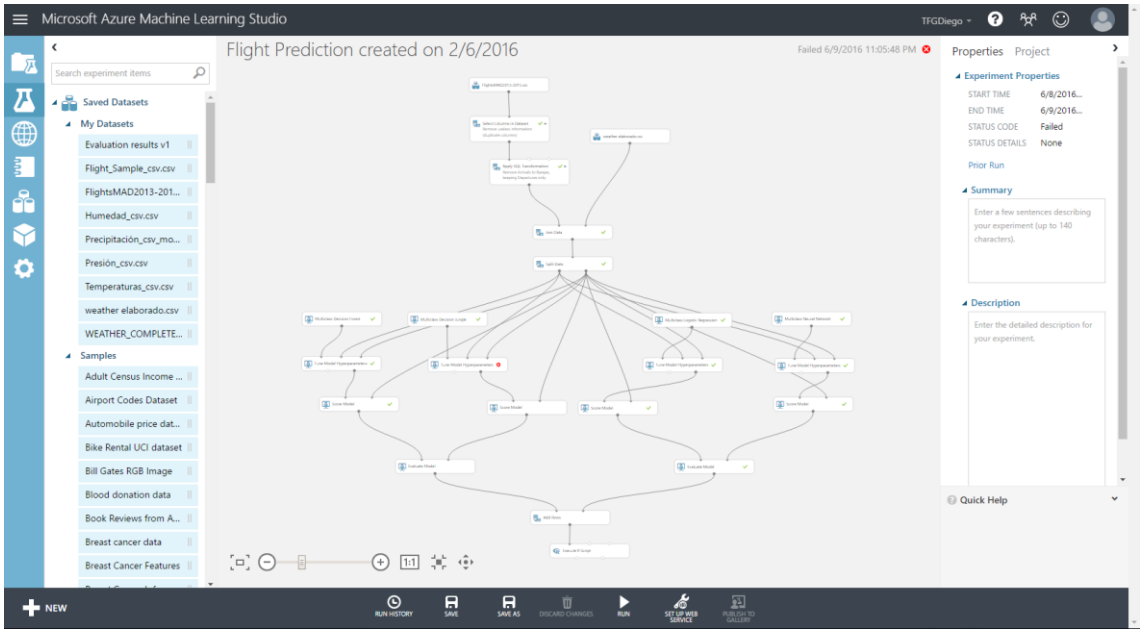


Figura 31. Experimento para la elección del algoritmo más efectivo.

Después de ejecutar el modelo creado para la elección del algoritmo más preciso, se decidió escoger el algoritmo Multiclass Decision Forest, ya que fue con el que se obtuvieron mejores resultados. La precisión media fue de un 85,82% y fue el algoritmo más acertado a la hora de predecir retraso o cancelación.

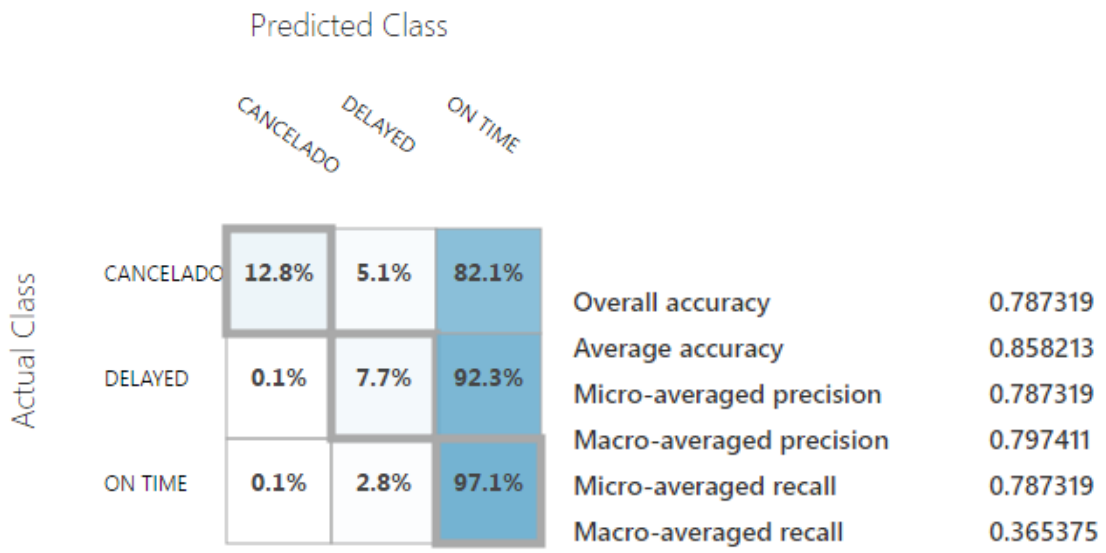


Figura 32. Evaluación del algoritmo elegido.

Despliegue del modelo

Por último, una vez obtenidos los datos y elegido el algoritmo, se realizó un nuevo experimento, sólo con el algoritmo elegido (el más preciso).

Una vez ejecutado, se creó un web service para poder acceder al modelo de una manera más sencilla, de forma similar a un applet, y se desplegó el modelo para poder aplicarlo a los datos de entrada a analizar.

En el siguiente apartado se realiza una descripción detallada del modelo predictivo final.

2.3.2. Descripción del modelo final

En la parte superior del modelo se encuentran los dos archivos históricos de datos, FlightsMAD2013-2015.csv y weather_elaborado.csv, que contienen la información de los vuelos y la meteorología respectivamente.

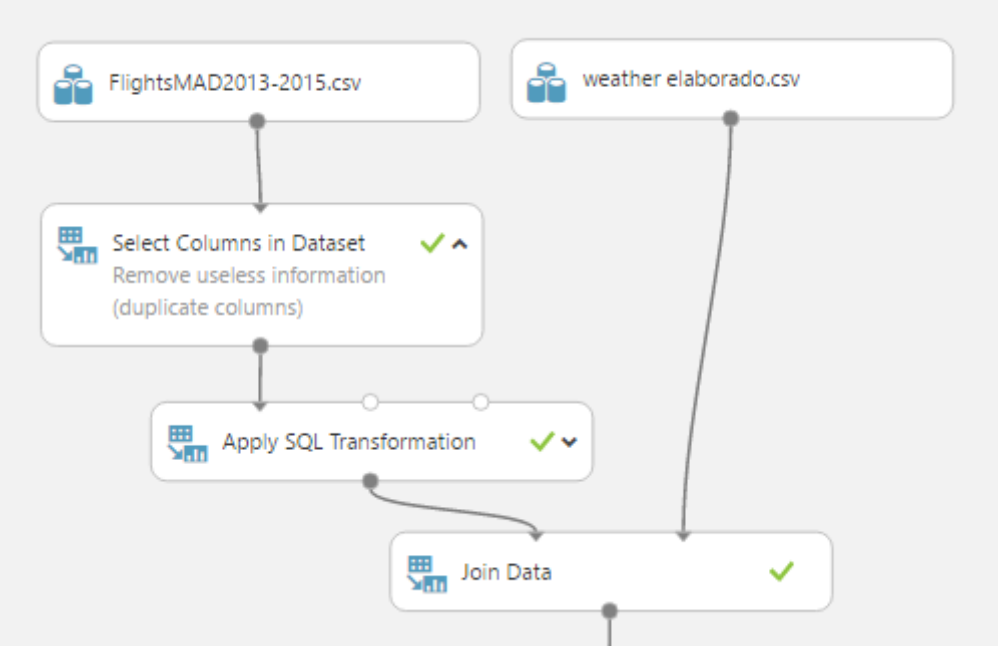


Figura 33. Pretratamiento de los datos, modelo final.

El archivo FlightsMAD2013-2015.csv pasa por dos bloques de tratamiento de datos. El primer bloque (Select Columns in Dataset) selecciona algunas columnas de información contenidas en el archivo original y las descarta (fechas y horas en formato incorrecto principalmente). El segundo bloque (Apply SQL Transformation) permite escribir código, que se usa para filtrar los vuelos, seleccionando únicamente las salidas desde Barajas y descartando las llegadas a Barajas.

El archivo weather_elaborado.csv ha sido tratado previamente en Excel, por lo que no necesita ningún cambio de formato ni filtro de información.

El siguiente paso es juntar los dos conjuntos de datos con el bloque Join Data.

Aquí concluye la etapa de pretratamiento de datos.

En la parte izquierda del modelo se encuentra el bloque correspondiente al algoritmo en el que se basa la predicción, Multiclass Decision Forest.

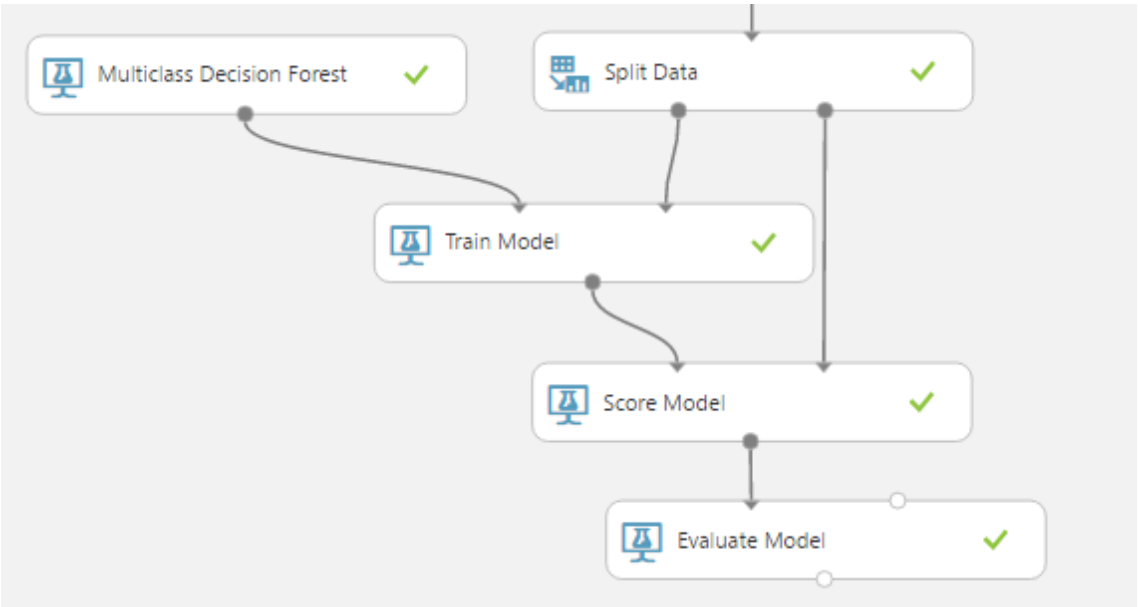


Figura 34. Algoritmo, entrenamiento y resultados del modelo final.

A continuación, se realiza el entrenamiento del modelo, que consiste en la ejecución del algoritmo sobre una fracción de los datos de que se dispone. Para tomar la fracción de los datos necesaria para el entrenamiento, se utiliza el bloque Split Data, que separa el total de los datos en dos subconjuntos de muestra aleatorios que contienen respectivamente el 80% y el 20% del total. Esta parte de los datos se utiliza junto con el algoritmo como entrada del siguiente bloque, Train Model. La salida de este bloque proporciona el modelo ya entrenado, que a su vez sirve como entrada para realizar una puntuación del modelo, con el bloque Score Model. Este bloque tiene como entrada el restante de los datos separados previamente.

Por último, y para comprobar el rendimiento del modelo, se añade el bloque Evaluate Model que proporciona los parámetros ya comentados de precisión del modelo y la matriz de confusión.

Aquí concluye la etapa de creación y entrenamiento del modelo.

El siguiente paso consiste en crear el web service para el modelo. Machine Learning Studio realiza esta operación de manera automática, y sólo hay que configurar dónde conectar la entrada y salida del web service. Después de este paso, el modelo se simplifica visualmente, quedando como se muestra en la figura.

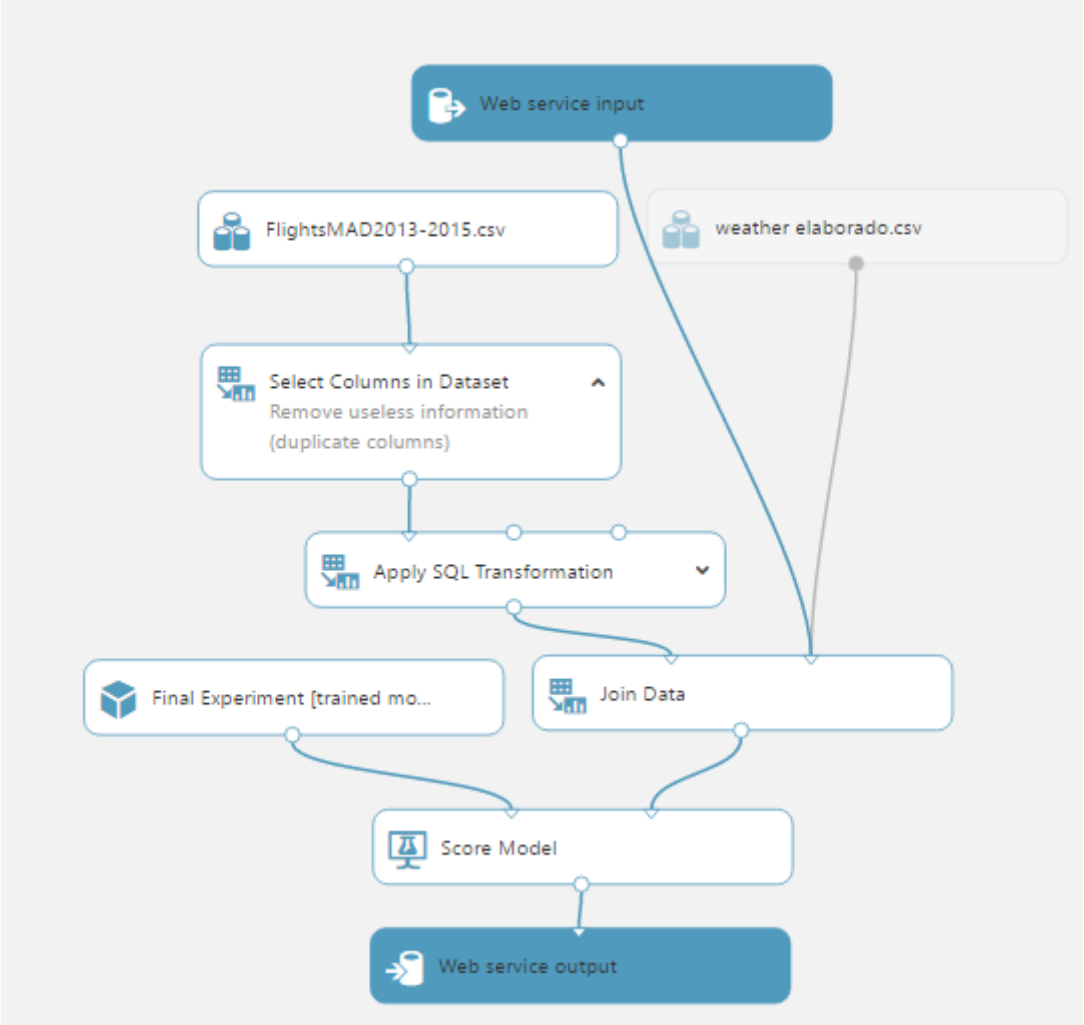


Figura 35. Modelo final incluyendo web service.

Después se vuelve a ejecutar el modelo con el web service ya implementado, y a continuación se puede realizar el despliegue del modelo. Una vez desplegado, el diagrama del modelo no cambia de aspecto, pero permite utilizar Excel como entrada de la consulta y salida de los datos de la predicción.

La hoja Excel tiene el aspecto de una hoja nueva normal, pero incluye una ventana lateral con el título Azure Machine Learning, que permite utilizar el web service.

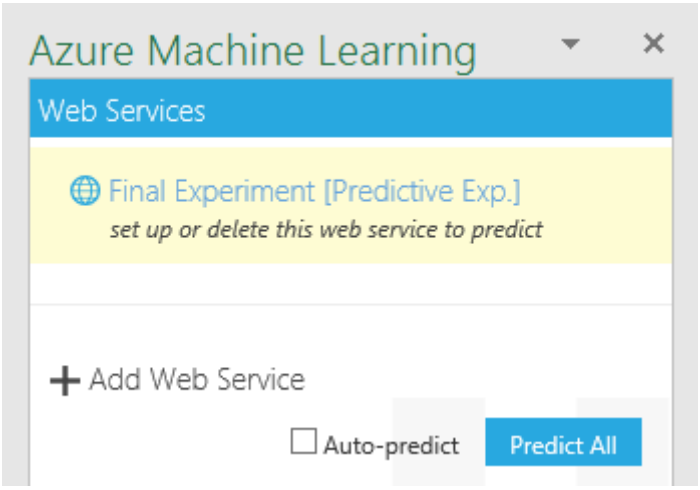


Figura 36. Azure Machine Learning web service.

Simplemente hay que escribir los datos de entrada (condiciones meteorológicas), señalar al web service dónde se encuentran estos datos, elegir dónde se escribirán los resultados, y hacer clic en 'Predecir'.

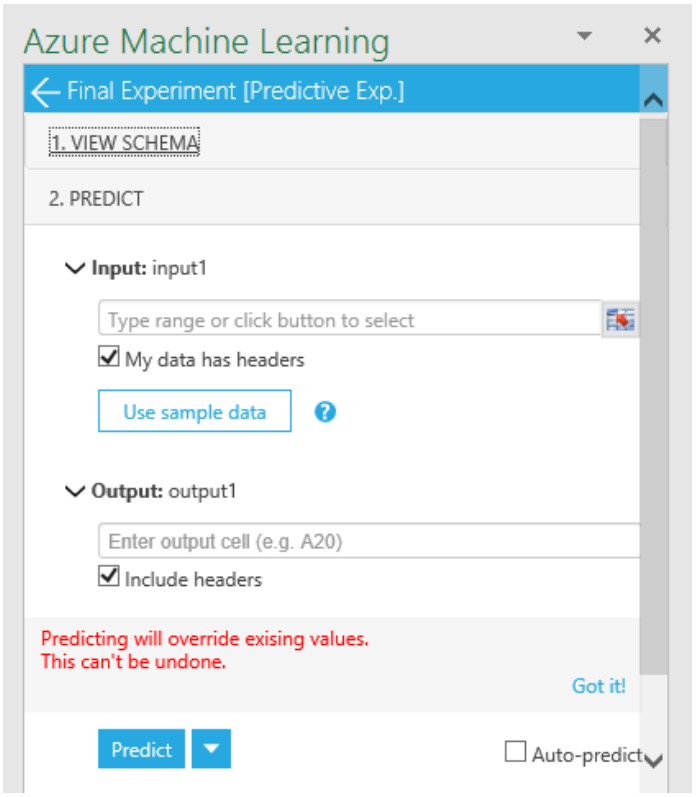


Figura 37. Web service en funcionamiento.

3. Resultados

3.1. Análisis de los resultados

Una vez finalizado el proyecto, se han obtenido los siguientes resultados:

El resultado más destacable que se ha obtenido es todo el sistema creado al completo. Se ha conseguido la amplitud deseada en el momento del enfoque del proyecto, y la implementación de todos los elementos que lo componen es necesaria para su correcto funcionamiento. Para analizarlos de forma sencilla, los resultados se pueden dividir en las siguientes partes:

- Se ha obtenido un sistema de toma de datos meteorológicos en tiempo real mediante un dispositivo de IoT.
- Se ha obtenido un entorno en la nube que permite la conexión del dispositivo IoT y la transmisión y almacenamiento de la información.
- Se ha obtenido un modelo integrado en la nube y conectado al dispositivo que elabora predicciones razonablemente fiables y precisas.
- Se ha obtenido un panel de control de PowerBI fácilmente reconfigurable que presenta los datos y los analiza en tiempo real.
- Se ha obtenido una hoja Excel que permite realizar consultas al modelo de forma sencilla.
- Se han definido una serie de futuras mejoras que se pueden implementar de forma sencilla en el proyecto.

3.2. Presentación de los datos

Uno de los objetivos del proyecto era la visualización de los datos utilizando PowerBI, de manera que la interpretación y análisis de los mismos fuera sencilla. A continuación, se incluye un panel realizado en PowerBI que facilita la visualización de los datos climáticos recogidos con la placa Sparkfun.

El panel creado incluye las principales medidas tomadas por la placa a lo largo del tiempo, y da libertad total al usuario para configurar su propio panel de visualización. Al

ser un ejemplo, se han incluido varios tipos de gráficos diferentes para ilustrar la variedad de opciones disponibles.

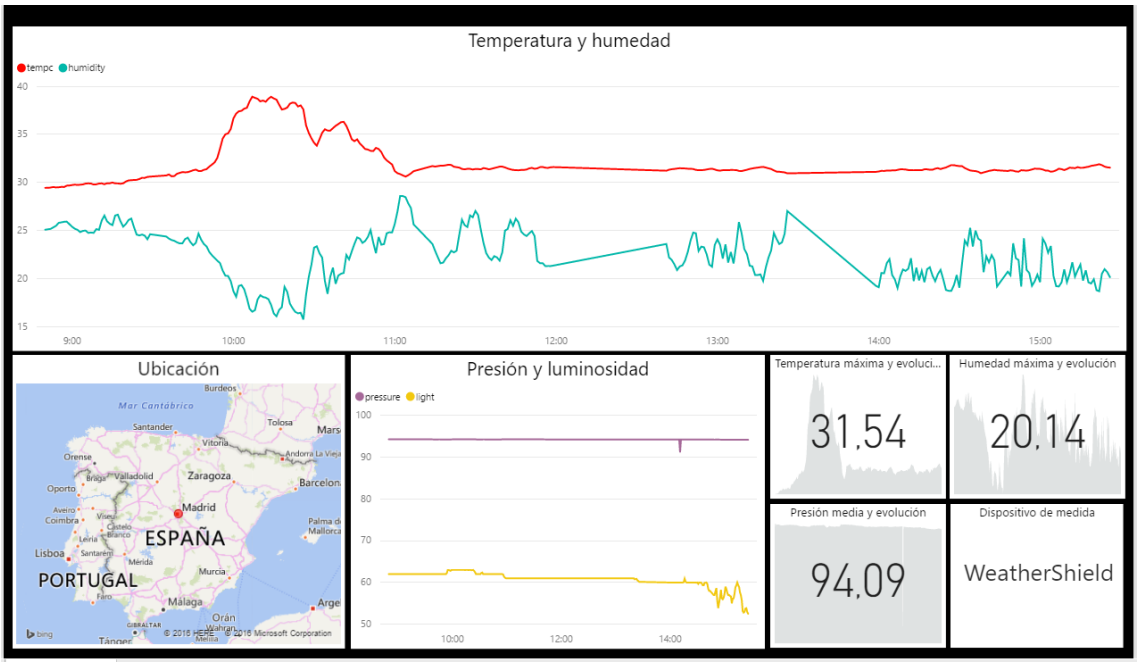


Figura 38. Panel de PowerBI.

PowerBI tiene varias opciones adicionales, como por ejemplo compartir el panel con otros usuarios mediante un link, o la excelente conectividad con Excel. Sin embargo, su característica más llamativa es la posibilidad de preguntar al propio panel acerca de los datos que contiene, y la capacidad de PowerBI para responder a la pregunta.

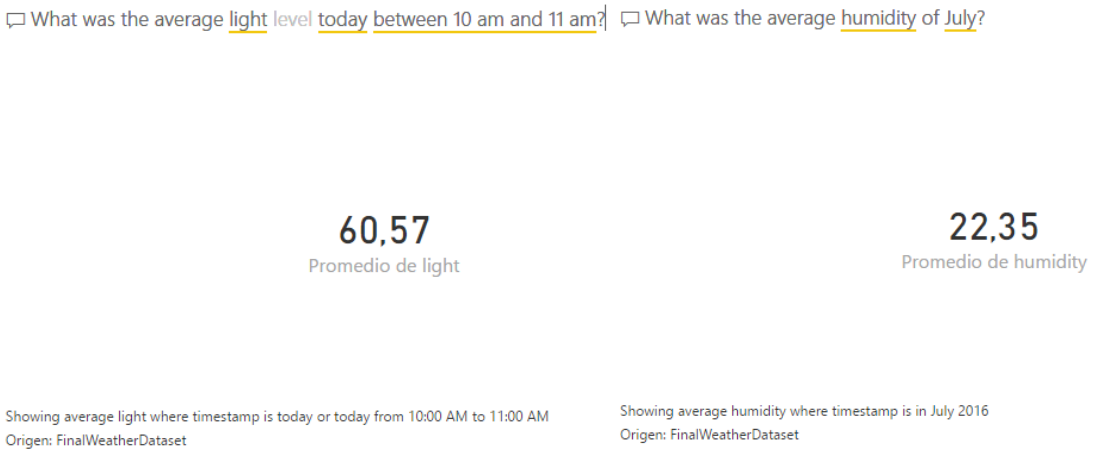


Figura 39. Preguntas y Respuestas, característica de PowerBI.

4. Conclusiones

Para concluir la memoria de este proyecto, se ha querido realizar un análisis de las tareas y objetivos cumplidos durante el desarrollo del mismo, así como una interpretación de los resultados obtenidos y una serie de posibles mejoras a implementar en el futuro.

Las tareas y objetivos cuya consecución se ha logrado durante el proyecto son las siguientes:

- Se ha logrado configurar la placa Sparkfun y el Weather Shield para la toma de datos meteorológicos en tiempo real.
- Se ha logrado obtener una frecuencia de medida y una precisión de los datos excelentes, adecuadas a las necesidades del modelo.
- Se ha logrado crear y configurar correctamente un entorno de trabajo en la nube
- Se ha logrado conectar correctamente el dispositivo de toma de datos con dicho entorno.
- Se ha logrado una transmisión de datos dispositivo-nube satisfactoria, y además se ha integrado la consolidación en la nube de los datos recibidos.
- Se ha logrado con éxito crear un modelo predictivo en la nube, completando todas las etapas habituales en este tipo de proyectos (tratamiento de los datos, elección del algoritmo y despliegue del modelo).
- Se ha comprobado la gran complejidad del modelo, y se han implementado las correcciones necesarias durante el proceso.
- En definitiva, se ha logrado con éxito realizar un proyecto ‘end-to-end’, cubriendo con suficiente detalle una gran variedad de tecnologías y recursos desde la captura inicial de los datos hasta la obtención y análisis de los resultados de la predicción.

A continuación, se presentan las conclusiones extraídas a partir del análisis de los resultados:

- Queda patente la gran complejidad de este tipo de modelos predictivos, no sólo por la cantidad de opciones a configurar durante la creación del mismo, sino también

por la constate necesidad de corrección de errores durante el proceso y los tiempos de ejecución del modelo.

- A la vista de los resultados, las condiciones meteorológicas son sólo uno de los factores que pueden causar retrasos y cancelaciones en los vuelos, y ésta es la principal causa de que el porcentaje de acierto en estos casos sea menor.

Por último, se presentan algunas de las posibles soluciones a considerar en el futuro:

- Para hacer la predicción más precisa, sería necesario disponer de información relativa a otras causas de cancelación y retraso, como por ejemplo fallos de gestión, retrasos de pasajeros, retrasos en vuelos anteriores, disponibilidad de pistas, fallos técnicos en las aeronaves, motivos burocráticos o políticos y muchos otros.
- La principal ventaja del formato de este modelo es la fácil implementación de nuevos datos, tras una simple fase inicial de pretratamiento y formato.
- Dada la gran capacidad de almacenamiento y análisis de los datos, podría buscarse una aplicación práctica de dicha capacidad para obtener una mayor información, no sólo acerca de los vuelos retrasados o cancelados sino cualquier tipo de información sobre todos los vuelos.
- Otra de las opciones a considerar en el futuro para mejorar la experiencia del usuario (destinatario final de la predicción), sería la creación de una aplicación móvil que implementase el modelo y presentase los resultados de una forma sencilla y fácil de interpretar.

5. Referencias

En la elaboración del proyecto se ha obtenido información de una serie de páginas web que se incluyen a continuación, junto con las referencias correspondientes:

[1] Sensor. En: Wikipedia, la enciclopedia libre [Internet]. 2016. Disponible en: https://es.wikipedia.org/w/index.php?title=Sensor&oldid=92240419

[2] Historia de los sensores [Internet]. Disponible en: http://electronica-general.blogspot.com.es/2013/11/historia-de-los-sensores_22.html

[3] Thyssen Krupp Elevator e IoT | Microsoft [Internet]. Disponible en: https://www.microsoft.com/es-es/server-cloud/customer-stories/thyssen-krupp-elevator.aspx

[4] SparkFun Weather Shield - DEV-12081 - SparkFun Electronics [Internet]. Disponible en: https://www.sparkfun.com/products/12081

[5] Internet de las cosas. En: Wikipedia, la enciclopedia libre [Internet]. 2016. Disponible en: https://es.wikipedia.org/w/index.php?title=Internet_de_las_cosas&oldid=92028779

[6] Microsoft Azure: plataforma y servicios de informática en la nube [Internet]. Disponible en: https://azure.microsoft.com/es-es/

[7] Connections Counter: The Internet of Everything in Motion [Internet]. Disponible en: https://newsroom.cisco.com/feature-content?type=webcontent&articleId=1208342

[8] Technology Research | Gartner Inc. [Internet]. Disponible en: http://www.gartner.com/technology/home.jsp

[9] Spain Weather - AccuWeather.com [Internet]. AccuWeather. Disponible en: http://www.accuweather.com/en/es/spain-weather

[10] Azure offers market leading support for SAP HANA workloads | Blog | Microsoft Azure [Internet]. Disponible en: https://azure.microsoft.com/es-es/blog/azure-offers-market-leading-support-for-sap-hana-workloads/

[11] Capabilities | SAP HANA [Internet]. Disponible en: https://hana.sap.com/capabilities.html

[12] What is Salesforce? Cloud CRM Solutions - Salesforce Europe [Internet]. Salesforce.com. Disponible en: http://www.salesforce.com/eu/crm/what-is-salesforce/

[13] Power BI | Herramientas de BI para la visualización de datos interactivos [Internet]. Disponible en: https://powerbi.microsoft.com/es-es/

[14] Análisis de transmisiones - Análisis de datos en tiempo real | Microsoft Azure [Internet]. Disponible en: https://azure.microsoft.com/es-es/services/stream-analytics/

[15] Características | Microsoft Power BI [Internet]. Disponible en: https://powerbi.microsoft.com/es-es/features/

[16] Centro de IoT de Azure | Microsoft Azure [Internet]. Disponible en: https://azure.microsoft.com/es-es/services/iot-hub/

[17] Cómo elegir un algoritmo en Aprendizaje automático de Azure | Microsoft Azure [Internet]. Disponible en: https://azure.microsoft.com/es-es/documentation/articles/machine-learning-algorithm-choice/

[18] Node.js - Connected Weather Station [Internet]. ThingLabs.io. Disponible en: /workshop/js/weather/

[19] Portal de Microsoft Azure | Microsoft Azure [Internet]. Disponible en: https://azure.microsoft.com/es-es/features/azure-portal/

[20] ¿Qué es el Estudio de aprendizaje automático de Azure? | Microsoft Azure [Internet]. Disponible en: https://azure.microsoft.com/es-es/documentation/articles/machine-learning-what-is-ml-studio/

6. Apéndices

Como apéndice, se añade una copia de la solicitud de los datos meteorológicos enviada a AEMET.

GOBIERNO DE ESPAÑA — MINISTERIO DE AGRICULTURA, ALIMENTACIÓN Y MEDIO AMBIENTE — AEMet Agencia Estatal de Meteorología

SOLICITUD DE PRESTACIONES METEOROLÓGICAS (L1)

1. DATOS DEL SOLICITANTE

CIF/NIF: [redacted]	Empresa (Nombre) // Particular (Nombre y Apellidos): DIEGO IRIBARREN BARO		
Su referencia:			
Sector de actividad(*): ESTUDIANTES / Estudiantes			
☐Empresa Privada	☐ Empresa Pública	☐ Administración Pública	☒ Particular/Autónomo
Domicilio Fiscal [redacted]			Código Postal: [redacted]
Localidad: MADRID	Provincia: Madrid		Pais: España
Telefono: [redacted]	Fax:	E-mail: [redacted]	

(*) En caso de administración pública o enseñanza universitaria, rellenar el apartado 5 y cumplimentar (1) para obtener el descuento aplicable en lel precio de la información y presentar documento original.

2. DATOS DE LA PERSONA DE CONTACTO (rellenar únicamente en caso de ser distintos que los del solicitante)

Persona de contacto (nombre y apellidos):		
Teléfono:	Fax:	E-mail:
Dirección de contacto:		

3. DESCRIPCIÓN DE LA PRESTACIÓN SOLICITADA

Proyecto de fin de grado universitario con fines académicos. Ver ficheros adjuntos.

Ficheros adjuntos: Solicitud AEMET.pdf.
Se han adjuntado a la solicitud ficheros con las estaciones y variables seleccionadas.

Si ha solicitado información de archivo ¿Necesita que se certifique? Si ☐ No ☒
¿Autoriza a que en caso de no existir información de las localidades o puntos solicitados se facilite la de los observatorios más proximos? Si ☒ No ☐

4. DATOS REFERIDOS AL SOPORTE Y MEDIO DE SUMINISTRO DE LA INFORMACIÓN

Soporte: ☐ Papel ☒ Informático

Medio: ☐ Correo ☐ Fax (según disponibilidad) ☒ Recogida en mano ☐ E-mail (solo ficheros) ☐ Otros (indique cual):

5. USO QUE SE VA HACER DE LA INFORMACIÓN (VOLUNTARIO)

Con el fin de poder facilitarle la información más adecuada, especifique la utilización que va a hacer de ella:
Entrenamiento del modelo de una prueba de concepto de Internet of Things

El firmante declara que los datos de esta solicitud son ciertos y acepta las obligaciones que figuran en el reverso que declara conocer.

(1) Organismo/ Universidad: Pontificia Comillas — Lugar, fecha y firma del solicitante
Departamento: Electrónica, Automática y Comunicaciones

Vº Bº Jefe Departamento
(Nombre, firma y sello)

Raúl Rodríguez Pecharromán

UNIVERSIDAD PONTIFICIA ICAI ICADE COMILLAS MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
DEPARTAMENTO DE ELECTRÓNICA Y AUTOMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MODELO PREDICTIVO. MACHINE LEARNING APLICADO AL ANÁLISIS DE DATOS CLIMÁTICOS CAPTURADOS POR UNA PLACA SPARKFUN.

Documento 2

PRESUPUESTO

Autor: Diego Iribarren Baró

Director: Jaime Pereña Pinedo

Índice del presupuesto

Índice de tablas

1. Mediciones

En este apartado se establecen los componentes que han sido necesarios para el desarrollo del proyecto. Estos componentes se han dividido en distintas categorías para facilitar el acceso a la información.

1.1. Componentes principales

Se consideran componentes principales los elementos que forman parte del sistema de captura de datos meteorológicos.

Componente	Marca	Cantidad
Placa Redboard	Sparkfun	1
Weather Shield	Sparkfun	1
Cable USB – mini-USB	Sparkfun	1

Tabla 1. Medición de componentes principales.

1.2. Equipo

En este apartado se incluyen los equipos utilizados a lo largo del proyecto.

Equipo	Horas de proyecto	Horas al año	Cantidad
Ordenador	500	2.000	1

Tabla 2. Medición de equipos.

1.3. Software

A continuación, se listan los programas utilizados durante el proyecto y las horas de uso de cada uno de ellos.

Programa	Horas de proyecto	Horas al año	Cantidad
Arduino	10	100	1
Visual Studio	20	100	1
Microsoft Word	200	1.000	1
Microsoft Excel	150	1.500	1
Microsoft PowerPoint	10	500	1
Microsoft PowerBI	50	250	1
Machine Learning Studio	350	700	1
Stream Analytics	100	300	1
IoT Hub	150	400	1

Tabla 3. Medición de software.

1.4. Mano de obra

Por último, se definen las horas destinadas a cada una de las tareas desarrolladas en el proyecto.

Tarea	**Horas**
Configuración de la placa	15
Creación del código	15
Configuración del entorno en la nube	40
Toma de datos	50
Tratamiento de los datos	50
Creación del modelo	70
Entrenamiento del modelo	60
Correcciones y ajustes	20
Redacción de documentos	250

Tabla 4. Medición de mano de obra.

2. Precios unitarios

En este apartado se establecen los precios unitarios referentes a los componentes, equipos, programas y tareas definidos en el apartado anterior.

2.1. Componentes principales

Componente	**Precio (€/ud.)**
Placa Redboard	19,95
Weather Shield	39,95
Cable USB – mini-USB	3,95

Tabla 5. Precios unitarios de los componentes principales.

2.2. Equipo

Equipo	**Precio (€/ud.)**
Ordenador	900

Tabla 6. Precios unitarios de los equipos.

2.3. Software

Programa	**Precio (€/ud.)**
Arduino	Software libre
Visual Studio	
Microsoft Word	150
Microsoft Excel	
Microsoft PowerPoint	
Microsoft PowerBI	243,61
Machine Learning Studio	
Stream Analytics	
IoT Hub	

Tabla 7. Precios unitarios del software.

2.4. Mano de obra

Tarea	**Precio (€/hora)**
Configuración de la placa	20
Creación del código	30
Configuración del entorno en la nube	40
Toma de datos	15
Tratamiento de los datos	20
Creación del modelo	60
Entrenamiento del modelo	60
Correcciones y ajustes	30
Redacción de documentos	20

Tabla 8. Precios unitarios de la mano de obra.

3. Sumas parciales

En este apartado se calculan los importes parciales de cada punto, tomando como base las mediciones y precios unitarios calculados anteriormente.

3.1. Componentes principales

Componente	**Cantidad**	**Precio unitario (€/ud.)**	**Coste total (€)**
Placa Redboard	1	19,95	19,95
Weather Shield	1	39,95	39,95
Cable USB – mini-USB	1	3,95	3,95
		Total	63,85 €
Sesenta y tres euros con ochenta y cinco céntimos			

Tabla 9. Sumas parciales de los componentes principales.

3.2. Equipo

Para el cálculo del coste de los equipos se usa la ecuación indicada a continuación, que incluye en el cálculo el impacto de las horas totales de uso y el porcentaje de amortización anual.

$$Coste(€) = \frac{Amort.\ anual}{100} \times \frac{Horas\ Proyecto}{Horas\ anuales} \times Precio\ total(€)$$

Equipo	**Horas de proyecto**	**Horas al año**	**Cantidad**	**Precio total (€)**	**Amortización anual (%)**	**Coste total (€)**
Ordenador	500	2000	1	900	33%	74,25
					Total	74,25 €
Setenta y cuatro euros con veinticinco céntimos						

Tabla 10. Sumas parciales de los equipos.

3.3. Software

Para el cálculo del coste del software se usa también la ecuación indicada a continuación, que incluye en el cálculo el impacto de las horas totales de uso y la amortización anual.

$$Coste(€) = \frac{Amort.\ anual}{100} \times \frac{Horas\ Proyecto}{Horas\ anuales} \times Precio\ total(€)$$

Programa	Horas de proyecto	Horas al año	Cantidad	Precio total (€)	Amortización anual (%)	Coste total (€)
Arduino	10	100	1	0	33%	0
Visual Studio	20	100	1	0	33%	0
Microsoft Word	200	1000	1	50	33%	3,3
Microsoft Excel	150	1500	1	50	33%	1,65
Microsoft PowerPoint	10	500	1	50	33%	0,33
Microsoft PowerBI	50	Pago por uso	1	243,61	Pago por uso	243,61
Machine Learning Studio	350		1			
Stream Analytics	100		1			
IoT Hub	150		1			
					Total	248,89 €
Doscientos cuarenta y ocho euros con ochenta y nueve céntimos						

Tabla 11. Sumas parciales del software.

3.4. Mano de obra

Tarea	Horas	Precio (€/hora)	Coste total (€)
Configuración de la placa	15	20	300
Creación del código	15	30	450
Configuración del entorno en la nube	40	40	1.600
Toma de datos	50	15	750
Tratamiento de los datos	50	20	1.000
Creación del modelo	70	60	4.200
Entrenamiento del modelo	60	60	3.600
Correcciones y ajustes	20	30	600
Redacción de documentos	200	20	4.000
		Total	16.500 €
Dieciséis mil quinientos euros			

Tabla 12. Sumas parciales de la mano de obra.

4. Presupuesto general

Por último, se suman todas las partidas anteriores y se obtiene el presupuesto total necesario para la realización del proyecto.

Hay que añadir además el coste de los datos solicitados a AEMET, que asciende a un total de 131,58€.

Partida	**Coste (€)**
Componentes principales	63,85
Equipo	74,25
Software	248,89
Mano de obra	16.500
Solicitud de los datos	131,58
Total	17.018,57
Diecisiete mil dieciocho euros con cincuenta y siete céntimos	

Tabla 13. Presupuesto general del proyecto.

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MODELO PREDICTIVO. MACHINE LEARNING APLICADO AL ANÁLISIS DE DATOS CLIMÁTICOS CAPTURADOS POR UNA PLACA SPARKFUN.

Anexo I

CÓDIGO FUENTE

Autor: Diego Iribarren Baró

Director: Jaime Pereña Pinedo

Índice del Anexo I: Código fuente

1. Actualización de la Firmata

La Firmata es un protocolo genérico de comunicación con controladores desde el software contenido en un ordenador. Para actualizarla hay que ejecutar el código obtenido en la placa Sparkfun usando el programa Arduino.

```
/*
 Firmata is a generic protocol for communicating with microcontrollers
 from software on a host computer. It is intended to work with
 any host computer software package.

 To download a host software package, please clink on the following link
 to open the list of Firmata client libraries your default browser.

 https://github.com/firmata/arduino#firmata-client-libraries

 Copyright (C) 2006-2008 Hans-Christoph Steiner.  All rights reserved.
 Copyright (C) 2010-2011 Paul Stoffregen.  All rights reserved.
 Copyright (C) 2009 Shigeru Kobayashi.  All rights reserved.
 Copyright (C) 2009-2015 Jeff Hoefs.  All rights reserved.

 This library is free software; you can redistribute it and/or
 modify it under the terms of the GNU Lesser General Public
 License as published by the Free Software Foundation; either
 version 2.1 of the License, or (at your option) any later version.

 See file LICENSE.txt for further informations on licensing terms.

 Last updated by Jeff Hoefs: November 7th, 2015
*/

#include <Servo.h>
#include <Wire.h>
#include <Firmata.h>

#define I2C_WRITE                   B00000000
#define I2C_READ                    B00001000
#define I2C_READ_CONTINUOUSLY       B00010000
#define I2C_STOP_READING            B00011000
#define I2C_READ_WRITE_MODE_MASK     B00011000
#define I2C_10BIT_ADDRESS_MODE_MASK B00100000
#define I2C_MAX_QUERIES             8
#define I2C_REGISTER_NOT_SPECIFIED  -1
```

```
// the minimum interval for sampling analog input
#define MINIMUM_SAMPLING_INTERVAL   1


/*==============================================================================
        GLOBAL VARIABLES
==============================================================================*/

/* analog inputs */
int analogInputsToReport = 0; // bitwise array to store pin reporting

/* digital input ports */
byte reportPINs[TOTAL_PORTS];       // 1 = report this port, 0 = silence
byte previousPINs[TOTAL_PORTS];     // previous 8 bits sent

/* pins configuration */
byte pinConfig[TOTAL_PINS];         // configuration of every pin
byte portConfigInputs[TOTAL_PORTS]; // each bit: 1 = pin in INPUT, 0 = anything else
int pinState[TOTAL_PINS];           // any value that has been written

/* timer variables */
unsigned long currentMillis;        // store the current value from millis()
unsigned long previousMillis;       // for comparison with currentMillis
unsigned int samplingInterval = 19; // how often to run the main loop (in ms)

/* i2c data */
struct i2c_device_info {
  byte addr;
  int reg;
  byte bytes;
};

/* for i2c read continuous more */
i2c_device_info query[I2C_MAX_QUERIES];

byte i2cRxData[32];
boolean isI2CEnabled = false;
signed char queryIndex = -1;
// default delay time between i2c read request and Wire.requestFrom()
unsigned int i2cReadDelayTime = 0;

Servo servos[MAX_SERVOS];
byte servoPinMap[TOTAL_PINS];
```

```
byte detachedServos[MAX_SERVOS];
byte detachedServoCount = 0;
byte servoCount = 0;

boolean isResetting = false;

/* utility functions */
void wireWrite(byte data)
{
#if ARDUINO >= 100
  Wire.write((byte)data);
#else
  Wire.send(data);
#endif
}

byte wireRead(void)
{
#if ARDUINO >= 100
  return Wire.read();
#else
  return Wire.receive();
#endif
}

/*============================================================================
   FUNCTIONS
============================================================================*/

void attachServo(byte pin, int minPulse, int maxPulse)
{
  if (servoCount < MAX_SERVOS) {
    // reuse indexes of detached servos until all have been reallocated
    if (detachedServoCount > 0) {
      servoPinMap[pin] = detachedServos[detachedServoCount - 1];
      if (detachedServoCount > 0) detachedServoCount--;
    } else {
      servoPinMap[pin] = servoCount;
      servoCount++;
    }
    if (minPulse > 0 && maxPulse > 0) {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin), minPulse, maxPulse);
    } else {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin));
```

```
    }
  } else {
    Firmata.sendString("Max servos attached");
  }
}

void detachServo (byte pin)
{
  servos[servoPinMap[pin]].detach();
  // if we're detaching the last servo, decrement the count
  // otherwise store the index of the detached servo
  if (servoPinMap[pin] == servoCount && servoCount > 0) {
    servoCount--;
  } else if (servoCount > 0) {
    // keep track of detached servos because we want to reuse their indexes
    // before incrementing the count of attached servos
    detachedServoCount++;
    detachedServos[detachedServoCount - 1] = servoPinMap[pin];
  }

  servoPinMap[pin] = 255;
}

void readAndReportData(byte address, int theRegister, byte numBytes) {
  // allow I2C requests that don't require a register read
  // for example, some devices using an interrupt pin to signify new data available
  // do not always require the register read so upon interrupt you call
Wire.requestFrom()
  if (theRegister != I2C_REGISTER_NOT_SPECIFIED) {
    Wire.beginTransmission(address);
    wireWrite((byte)theRegister);
    Wire.endTransmission();
    // do not set a value of 0
    if (i2cReadDelayTime > 0) {
      // delay is necessary for some devices such as WiiNunchuck
      delayMicroseconds(i2cReadDelayTime);
    }
  } else {
    theRegister = 0;  // fill the register with a dummy value
  }

  Wire.requestFrom(address, numBytes);  // all bytes are returned in requestFrom

  // check to be sure correct number of bytes were returned by slave
```

```
  if (numBytes < Wire.available()) {
    Firmata.sendString("I2C: Too many bytes received");
  } else if (numBytes > Wire.available()) {
    Firmata.sendString("I2C: Too few bytes received");
  }

  i2cRxData[0] = address;
  i2cRxData[1] = theRegister;

  for (int i = 0; i < numBytes && Wire.available(); i++) {
    i2cRxData[2 + i] = wireRead();
  }

  // send slave address, register and received bytes
  Firmata.sendSysex(SYSEX_I2C_REPLY, numBytes + 2, i2cRxData);
}

void outputPort(byte portNumber, byte portValue, byte forceSend)
{
  // pins not configured as INPUT are cleared to zeros
  portValue = portValue & portConfigInputs[portNumber];
  // only send if the value is different than previously sent
  if (forceSend || previousPINs[portNumber] != portValue) {
    Firmata.sendDigitalPort(portNumber, portValue);
    previousPINs[portNumber] = portValue;
  }
}

/* -----------------------------------------------------------------------------
   check all the active digital inputs for change of state, then add any events
   to the Serial output queue using Serial.print() */
void checkDigitalInputs(void)
{
  /* Using non-looping code allows constants to be given to readPort().
     The compiler will apply substantial optimizations if the inputs
     to readPort() are compile-time constants. */
  if (TOTAL_PORTS > 0 && reportPINs[0]) outputPort(0, readPort(0, portConfigInputs[0]),
false);
  if (TOTAL_PORTS > 1 && reportPINs[1]) outputPort(1, readPort(1, portConfigInputs[1]),
false);
  if (TOTAL_PORTS > 2 && reportPINs[2]) outputPort(2, readPort(2, portConfigInputs[2]),
false);
  if (TOTAL_PORTS > 3 && reportPINs[3]) outputPort(3, readPort(3, portConfigInputs[3]),
false);
```

```
  if (TOTAL_PORTS > 4 && reportPINs[4]) outputPort(4, readPort(4, portConfigInputs[4]),
false);
  if (TOTAL_PORTS > 5 && reportPINs[5]) outputPort(5, readPort(5, portConfigInputs[5]),
false);
  if (TOTAL_PORTS > 6 && reportPINs[6]) outputPort(6, readPort(6, portConfigInputs[6]),
false);
  if (TOTAL_PORTS > 7 && reportPINs[7]) outputPort(7, readPort(7, portConfigInputs[7]),
false);
  if (TOTAL_PORTS > 8 && reportPINs[8]) outputPort(8, readPort(8, portConfigInputs[8]),
false);
  if (TOTAL_PORTS > 9 && reportPINs[9]) outputPort(9, readPort(9, portConfigInputs[9]),
false);
  if (TOTAL_PORTS > 10 && reportPINs[10]) outputPort(10, readPort(10,
portConfigInputs[10]), false);
  if (TOTAL_PORTS > 11 && reportPINs[11]) outputPort(11, readPort(11,
portConfigInputs[11]), false);
  if (TOTAL_PORTS > 12 && reportPINs[12]) outputPort(12, readPort(12,
portConfigInputs[12]), false);
  if (TOTAL_PORTS > 13 && reportPINs[13]) outputPort(13, readPort(13,
portConfigInputs[13]), false);
  if (TOTAL_PORTS > 14 && reportPINs[14]) outputPort(14, readPort(14,
portConfigInputs[14]), false);
  if (TOTAL_PORTS > 15 && reportPINs[15]) outputPort(15, readPort(15,
portConfigInputs[15]), false);
}

// -----------------------------------------------------------------------------------------------
/* sets the pin mode to the correct state and sets the relevant bits in the
   two bit-arrays that track Digital I/O and PWM status
 */
void setPinModeCallback(byte pin, int mode)
{
  if (pinConfig[pin] == PIN_MODE_IGNORE)
    return;

  if (pinConfig[pin] == PIN_MODE_I2C && isI2CEnabled && mode != PIN_MODE_I2C) {
    // disable i2c so pins can be used for other functions
    // the following if statements should reconfigure the pins properly
    disableI2CPins();
  }
  if (IS_PIN_DIGITAL(pin) && mode != PIN_MODE_SERVO) {
    if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
      detachServo(pin);
    }
```

```
  }
  if (IS_PIN_ANALOG(pin)) {
    reportAnalogCallback(PIN_TO_ANALOG(pin), mode == PIN_MODE_ANALOG ? 1 : 0);
// turn on/off reporting
  }
  if (IS_PIN_DIGITAL(pin)) {
    if (mode == INPUT || mode == PIN_MODE_PULLUP) {
      portConfigInputs[pin / 8] |= (1 << (pin & 7));
    } else {
      portConfigInputs[pin / 8] &= ~(1 << (pin & 7));
    }
  }
  pinState[pin] = 0;
  switch (mode) {
    case PIN_MODE_ANALOG:
      if (IS_PIN_ANALOG(pin)) {
        if (IS_PIN_DIGITAL(pin)) {
          pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
          // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
          digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        }
        pinConfig[pin] = PIN_MODE_ANALOG;
      }
      break;
    case INPUT:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
        // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
        digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        pinConfig[pin] = INPUT;
      }
      break;
    case PIN_MODE_PULLUP:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT_PULLUP);
        pinConfig[pin] = PIN_MODE_PULLUP;
        pinState[pin] = 1;
      }
      break;
    case OUTPUT:
```

```
    if (IS_PIN_DIGITAL(pin)) {
      digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable PWM
      pinMode(PIN_TO_DIGITAL(pin), OUTPUT);
      pinConfig[pin] = OUTPUT;
    }
    break;
  case PIN_MODE_PWM:
    if (IS_PIN_PWM(pin)) {
      pinMode(PIN_TO_PWM(pin), OUTPUT);
      analogWrite(PIN_TO_PWM(pin), 0);
      pinConfig[pin] = PIN_MODE_PWM;
    }
    break;
  case PIN_MODE_SERVO:
    if (IS_PIN_DIGITAL(pin)) {
      pinConfig[pin] = PIN_MODE_SERVO;
      if (servoPinMap[pin] == 255 || !servos[servoPinMap[pin]].attached()) {
        // pass -1 for min and max pulse values to use default values set
        // by Servo library
        attachServo(pin, -1, -1);
      }
    }
    break;
  case PIN_MODE_I2C:
    if (IS_PIN_I2C(pin)) {
      // mark the pin as i2c
      // the user must call I2C_CONFIG to enable I2C for a device
      pinConfig[pin] = PIN_MODE_I2C;
    }
    break;
  default:
    Firmata.sendString("Unknown pin mode"); // TODO: put error msgs in EEPROM
 }
 // TODO: save status to EEPROM here, if changed
}

/*
  Sets the value of an individual pin. Useful if you want to set a pin value but
  are not tracking the digital port state.
  Can only be used on pins configured as OUTPUT.
  Cannot be used to enable pull-ups on Digital INPUT pins.
 */
void setPinValueCallback(byte pin, int value)
{
```

```
  if (pin < TOTAL_PINS && IS_PIN_DIGITAL(pin)) {
    if (pinConfig[pin] == OUTPUT) {
      pinState[pin] = value;
      digitalWrite(PIN_TO_DIGITAL(pin), value);
    }
  }
}

void analogWriteCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS) {
    switch (pinConfig[pin]) {
      case PIN_MODE_SERVO:
        if (IS_PIN_DIGITAL(pin))
          servos[servoPinMap[pin]].write(value);
        pinState[pin] = value;
        break;
      case PIN_MODE_PWM:
        if (IS_PIN_PWM(pin))
          analogWrite(PIN_TO_PWM(pin), value);
        pinState[pin] = value;
        break;
    }
  }
}

void digitalWriteCallback(byte port, int value)
{
  byte pin, lastPin, pinValue, mask = 1, pinWriteMask = 0;

  if (port < TOTAL_PORTS) {
    // create a mask of the pins on this port that are writable.
    lastPin = port * 8 + 8;
    if (lastPin > TOTAL_PINS) lastPin = TOTAL_PINS;
    for (pin = port * 8; pin < lastPin; pin++) {
      // do not disturb non-digital pins (eg, Rx & Tx)
      if (IS_PIN_DIGITAL(pin)) {
        // do not touch pins in PWM, ANALOG, SERVO or other modes
        if (pinConfig[pin] == OUTPUT || pinConfig[pin] == INPUT) {
          pinValue = ((byte)value & mask) ? 1 : 0;
          if (pinConfig[pin] == OUTPUT) {
            pinWriteMask |= mask;
          } else if (pinConfig[pin] == INPUT && pinValue == 1 && pinState[pin] != 1) {
            // only handle INPUT here for backwards compatibility
```

```
#if ARDUINO > 100
        pinMode(pin, INPUT_PULLUP);
#else
        // only write to the INPUT pin to enable pullups if Arduino v1.0.0 or earlier
        pinWriteMask |= mask;
#endif
       }
       pinState[pin] = pinValue;
      }
     }
     mask = mask << 1;
   }
   writePort(port, (byte)value, pinWriteMask);
 }
}


// -----------------------------------------------------------------------------
/* sets bits in a bit array (int) to toggle the reporting of the analogIns
 */
//void FirmataClass::setAnalogPinReporting(byte pin, byte state) {
//}
void reportAnalogCallback(byte analogPin, int value)
{
 if (analogPin < TOTAL_ANALOG_PINS) {
  if (value == 0) {
   analogInputsToReport = analogInputsToReport & ~ (1 << analogPin);
  } else {
   analogInputsToReport = analogInputsToReport | (1 << analogPin);
   // prevent during system reset or all analog pin values will be reported
   // which may report noise for unconnected analog pins
   if (!isResetting) {
    // Send pin value immediately. This is helpful when connected via
    // ethernet, wi-fi or bluetooth so pin states can be known upon
    // reconnecting.
    Firmata.sendAnalog(analogPin, analogRead(analogPin));
   }
  }
 }
 // TODO: save status to EEPROM here, if changed
}

void reportDigitalCallback(byte port, int value)
{
```

```
  if (port < TOTAL_PORTS) {
    reportPINs[port] = (byte)value;
    // Send port value immediately. This is helpful when connected via
    // ethernet, wi-fi or bluetooth so pin states can be known upon
    // reconnecting.
    if (value) outputPort(port, readPort(port, portConfigInputs[port]), true);
  }
  // do not disable analog reporting on these 8 pins, to allow some
  // pins used for digital, others analog.  Instead, allow both types
  // of reporting to be enabled, but check if the pin is configured
  // as analog when sampling the analog inputs.  Likewise, while
  // scanning digital pins, portConfigInputs will mask off values from any
  // pins configured as analog
}

/*==============================================================================
  SYSEX-BASED commands
==============================================================================*/

void sysexCallback(byte command, byte argc, byte *argv)
{
  byte mode;
  byte slaveAddress;
  byte data;
  int slaveRegister;
  unsigned int delayTime;

  switch (command) {
    case I2C_REQUEST:
      mode = argv[1] & I2C_READ_WRITE_MODE_MASK;
      if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {
        Firmata.sendString("10-bit addressing not supported");
        return;
      }
      else {
        slaveAddress = argv[0];
      }

      switch (mode) {
        case I2C_WRITE:
          Wire.beginTransmission(slaveAddress);
          for (byte i = 2; i < argc; i += 2) {
            data = argv[i] + (argv[i + 1] << 7);
            wireWrite(data);
```

```
  }
  Wire.endTransmission();
  delayMicroseconds(70);
  break;
 case I2C_READ:
  if (argc == 6) {
   // a slave register is specified
   slaveRegister = argv[2] + (argv[3] << 7);
   data = argv[4] + (argv[5] << 7);  // bytes to read
  }
  else {
   // a slave register is NOT specified
   slaveRegister = I2C_REGISTER_NOT_SPECIFIED;
   data = argv[2] + (argv[3] << 7);  // bytes to read
  }
  readAndReportData(slaveAddress, (int)slaveRegister, data);
  break;
 case I2C_READ_CONTINUOUSLY:
  if ((queryIndex + 1) >= I2C_MAX_QUERIES) {
   // too many queries, just ignore
   Firmata.sendString("too many queries");
   break;
  }
  if (argc == 6) {
   // a slave register is specified
   slaveRegister = argv[2] + (argv[3] << 7);
   data = argv[4] + (argv[5] << 7);  // bytes to read
  }
  else {
   // a slave register is NOT specified
   slaveRegister = (int)I2C_REGISTER_NOT_SPECIFIED;
   data = argv[2] + (argv[3] << 7);  // bytes to read
  }
  queryIndex++;
  query[queryIndex].addr = slaveAddress;
  query[queryIndex].reg = slaveRegister;
  query[queryIndex].bytes = data;
  break;
 case I2C_STOP_READING:
  byte queryIndexToSkip;
  // if read continuous mode is enabled for only 1 i2c device, disable
  // read continuous reporting for that device
  if (queryIndex <= 0) {
   queryIndex = -1;
```

```
      } else {
        // if read continuous mode is enabled for multiple devices,
        // determine which device to stop reading and remove its data from
        // the array, shifting other array data to fill the space
        for (byte i = 0; i < queryIndex + 1; i++) {
          if (query[i].addr == slaveAddress) {
            queryIndexToSkip = i;
            break;
          }
        }

        for (byte i = queryIndexToSkip; i < queryIndex + 1; i++) {
          if (i < I2C_MAX_QUERIES) {
            query[i].addr = query[i + 1].addr;
            query[i].reg = query[i + 1].reg;
            query[i].bytes = query[i + 1].bytes;
          }
        }
        queryIndex--;
      }
      break;
    default:
      break;
  }
  break;
case I2C_CONFIG:
  delayTime = (argv[0] + (argv[1] << 7));

  if (delayTime > 0) {
    i2cReadDelayTime = delayTime;
  }

  if (!isI2CEnabled) {
    enableI2CPins();
  }

  break;
case SERVO_CONFIG:
  if (argc > 4) {
    // these vars are here for clarity, they'll optimized away by the compiler
    byte pin = argv[0];
    int minPulse = argv[1] + (argv[2] << 7);
    int maxPulse = argv[3] + (argv[4] << 7);
```

```
    if (IS_PIN_DIGITAL(pin)) {
      if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
        detachServo(pin);
      }
      attachServo(pin, minPulse, maxPulse);
      setPinModeCallback(pin, PIN_MODE_SERVO);
    }
  }
  break;
case SAMPLING_INTERVAL:
  if (argc > 1) {
    samplingInterval = argv[0] + (argv[1] << 7);
    if (samplingInterval < MINIMUM_SAMPLING_INTERVAL) {
      samplingInterval = MINIMUM_SAMPLING_INTERVAL;
    }
  } else {
    //Firmata.sendString("Not enough data");
  }
  break;
case EXTENDED_ANALOG:
  if (argc > 1) {
    int val = argv[1];
    if (argc > 2) val |= (argv[2] << 7);
    if (argc > 3) val |= (argv[3] << 14);
    analogWriteCallback(argv[0], val);
  }
  break;
case CAPABILITY_QUERY:
  Firmata.write(START_SYSEX);
  Firmata.write(CAPABILITY_RESPONSE);
  for (byte pin = 0; pin < TOTAL_PINS; pin++) {
    if (IS_PIN_DIGITAL(pin)) {
      Firmata.write((byte)INPUT);
      Firmata.write(1);
      Firmata.write((byte)PIN_MODE_PULLUP);
      Firmata.write(1);
      Firmata.write((byte)OUTPUT);
      Firmata.write(1);
    }
    if (IS_PIN_ANALOG(pin)) {
      Firmata.write(PIN_MODE_ANALOG);
      Firmata.write(10); // 10 = 10-bit resolution
    }
    if (IS_PIN_PWM(pin)) {
```

```
        Firmata.write(PIN_MODE_PWM);
        Firmata.write(8); // 8 = 8-bit resolution
       }
       if (IS_PIN_DIGITAL(pin)) {
        Firmata.write(PIN_MODE_SERVO);
        Firmata.write(14);
       }
       if (IS_PIN_I2C(pin)) {
        Firmata.write(PIN_MODE_I2C);
        Firmata.write(1);  // TODO: could assign a number to map to SCL or SDA
       }
       Firmata.write(127);
     }
     Firmata.write(END_SYSEX);
     break;
   case PIN_STATE_QUERY:
     if (argc > 0) {
       byte pin = argv[0];
       Firmata.write(START_SYSEX);
       Firmata.write(PIN_STATE_RESPONSE);
       Firmata.write(pin);
       if (pin < TOTAL_PINS) {
         Firmata.write((byte)pinConfig[pin]);
         Firmata.write((byte)pinState[pin] & 0x7F);
         if (pinState[pin] & 0xFF80) Firmata.write((byte)(pinState[pin] >> 7) & 0x7F);
         if (pinState[pin] & 0xC000) Firmata.write((byte)(pinState[pin] >> 14) & 0x7F);
       }
       Firmata.write(END_SYSEX);
     }
     break;
   case ANALOG_MAPPING_QUERY:
     Firmata.write(START_SYSEX);
     Firmata.write(ANALOG_MAPPING_RESPONSE);
     for (byte pin = 0; pin < TOTAL_PINS; pin++) {
       Firmata.write(IS_PIN_ANALOG(pin) ? PIN_TO_ANALOG(pin) : 127);
     }
     Firmata.write(END_SYSEX);
     break;
  }
}

void enableI2CPins()
{
  byte i;
```

```
  // is there a faster way to do this? would probaby require importing
  // Arduino.h to get SCL and SDA pins
  for (i = 0; i < TOTAL_PINS; i++) {
    if (IS_PIN_I2C(i)) {
      // mark pins as i2c so they are ignore in non i2c data requests
      setPinModeCallback(i, PIN_MODE_I2C);
    }
  }

  isI2CEnabled = true;

  Wire.begin();
}

/* disable the i2c pins so they can be used for other functions */
void disableI2CPins() {
  isI2CEnabled = false;
  // disable read continuous mode for all devices
  queryIndex = -1;
}

/*==============================================================================
  SETUP()
==============================================================================*/

void systemResetCallback()
{
  isResetting = true;

  // initialize a defalt state
  // TODO: option to load config from EEPROM instead of default

  if (isI2CEnabled) {
    disableI2CPins();
  }

  for (byte i = 0; i < TOTAL_PORTS; i++) {
    reportPINs[i] = false;    // by default, reporting off
    portConfigInputs[i] = 0;  // until activated
    previousPINs[i] = 0;
  }

  for (byte i = 0; i < TOTAL_PINS; i++) {
    // pins with analog capability default to analog input
```

```
    // otherwise, pins default to digital output
    if (IS_PIN_ANALOG(i)) {
      // turns off pullup, configures everything
      setPinModeCallback(i, PIN_MODE_ANALOG);
    } else if (IS_PIN_DIGITAL(i)) {
      // sets the output to 0, configures portConfigInputs
      setPinModeCallback(i, OUTPUT);
    }

    servoPinMap[i] = 255;
  }
  // by default, do not report any analog inputs
  analogInputsToReport = 0;

  detachedServoCount = 0;
  servoCount = 0;

  /* send digital inputs to set the initial state on the host computer,
     since once in the loop(), this firmware will only send on change */
  /*
  TODO: this can never execute, since no pins default to digital input
        but it will be needed when/if we support EEPROM stored config
  for (byte i=0; i < TOTAL_PORTS; i++) {
    outputPort(i, readPort(i, portConfigInputs[i]), true);
  }
  */
  isResetting = false;
}

void setup()
{
  Firmata.setFirmwareVersion(FIRMATA_MAJOR_VERSION,
FIRMATA_MINOR_VERSION);

  Firmata.attach(ANALOG_MESSAGE, analogWriteCallback);
  Firmata.attach(DIGITAL_MESSAGE, digitalWriteCallback);
  Firmata.attach(REPORT_ANALOG, reportAnalogCallback);
  Firmata.attach(REPORT_DIGITAL, reportDigitalCallback);
  Firmata.attach(SET_PIN_MODE, setPinModeCallback);
  Firmata.attach(SET_DIGITAL_PIN_VALUE, setPinValueCallback);
  Firmata.attach(START_SYSEX, sysexCallback);
  Firmata.attach(SYSTEM_RESET, systemResetCallback);

  // to use a port other than Serial, such as Serial1 on an Arduino Leonardo or Mega,
```

```
  // Call begin(baud) on the alternate serial port and pass it to Firmata to begin like this:
  // Serial1.begin(57600);
  // Firmata.begin(Serial1);
  // then comment out or remove lines 701 - 704 below

  Firmata.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Only needed for ATmega32u4-based boards (Leonardo, etc).
  }
  systemResetCallback();  // reset to default config
}

/*==============================================================================
  LOOP()
==============================================================================*/
void loop()
{
  byte pin, analogPin;

  /* DIGITALREAD - as fast as possible, check for changes and output them to the
   * FTDI buffer using Serial.print()  */
  checkDigitalInputs();

  /* STREAMREAD - processing incoming messagse as soon as possible, while still
   * checking digital inputs.  */
  while (Firmata.available())
    Firmata.processInput();

  // TODO - ensure that Stream buffer doesn't go over 60 bytes

  currentMillis = millis();
  if (currentMillis - previousMillis > samplingInterval) {
    previousMillis += samplingInterval;
    /* ANALOGREAD - do all analogReads() at the configured sampling interval */
    for (pin = 0; pin < TOTAL_PINS; pin++) {
      if (IS_PIN_ANALOG(pin) && pinConfig[pin] == PIN_MODE_ANALOG) {
        analogPin = PIN_TO_ANALOG(pin);
        if (analogInputsToReport & (1 << analogPin)) {
          Firmata.sendAnalog(analogPin, analogRead(analogPin));
        }
      }
    }
    // report i2c data for all device with read continuous mode enabled
```

```
    if (queryIndex > -1) {
      for (byte i = 0; i < queryIndex + 1; i++) {
        readAndReportData(query[i].addr, query[i].reg, query[i].bytes);
      }
    }
  }
}
```

2. Package.json

Este archivo permite usar NPM para descargar los módulos Node necesarios para operar con la placa Sparkfun. Se han configurado las conexiones y dependencias para usar la versión más actualizada por defecto en la comunicación http.

```
{
    "name": "IoT-Labs-Arduino",
    "repository": {
      "type": "git",
      "url": "https://github.com/ThingLabsIo/IoTLabs/tree/master/Arduino/Weather"
    },
    "version": "0.1.0",
    "private":true,
    "description": "Sample app that connects a device to Azure using Node.js",
    "main": "weather.js",
    "author": "Diego Iribarren Baro",
    "license": "MIT",
    "dependencies": {
      "azure-iot-device": "latest",
      "azure-iot-device-amqp": "latest",
      "azure-iot-device-http": "latest",
      "johnny-five": "latest",
      "j5-sparkfun-weather-shield": "latest"
    }
  }
```

3. Weather.js

Este archivo contiene el código que la placa utiliza para tomar las medidas de los parámetros meteorológicos. Es un JavaScript que determina, entre otros parámetros, la frecuencia a la que se miden los datos meteorológicos, las variables en las que se almacenan los valores y los tipos de dichas variables.

Además, en él se configura también la conexión de la placa al Azure IoT Hub creado en la nube.

```
// https://github.com/ThingLabsIo/IoTLabs/tree/master/Arduino/Weather
'use strict';
// Define the objects you will be working with
var five = require ("johnny-five");
var Shield = require("j5-sparkfun-weather-shield")(five);
var device = require('azure-iot-device');

// Use factory function from AMQP-specific package
// Other options include HTTP (azure-iot-device-http) and MQTT (azure-iot-device-mqtt)
var clientFromConnectionString = require('azure-iot-device-http').clientFromConnectionString;

var location = process.env.DEVICE_LOCATION || 'MADRID';
var connectionString = process.env.IOTHUB_CONN || 'HostName=TFGDiego.azure-devices.net;DeviceId=WeatherShield;SharedAccessKey=';

// Create an Azure IoT client that will manage the connection to your IoT Hub
// The client is created in the context of an Azure IoT device, which is why
// you use a device-specific connection string.
var client = clientFromConnectionString(connectionString);
```

```
var deviceId = device.ConnectionString.parse(connectionString).DeviceId;

// Create a Johnny-Five board instance to represent your Particle Photon
// Board is simply an abstraction of the physical hardware, whether is is a
// Photon, Arduino, Raspberry Pi or other boards.
var board = new five.Board();

/*
// You may optionally specify the port by providing it as a property
// of the options object parameter. * Denotes system specific
// enumeration value (ie. a number)
// Windows
var board = new five.Board({ port: "COM*" });
*/

//This is where the program "really" starts.

// The board.on() executes the anonymous function when the
// board reports back that it is initialized and ready.
board.on("ready", function() {
    console.log("Your Weather Device is now connected.");

    // The SparkFun Weather Shield has two sensors on the I2C bus -
    // a humidity sensor (HTU21D) which can provide both humidity and temperature,
and a
    // barometer (MPL3115A2) which can provide both barometric pressure and
humidity.
    // Controllers for these are wrapped in a convenient plugin class:
    var weather = new Shield({
```

```
    variant: "ARDUINO", // or variant: "PHOTON",
    freq: 1000,       // [milisecs] Set the callback frequency to 1-second
    elevation: 648    // [meters] Go to http://www.WhatIsMyElevation.com to get your current elevation
    });

    // The weather.on("data", callback) function invokes the anonymous callback function
    // whenever the data from the sensor changes (no faster than every 25ms). The anonymous
    // function is scoped to the object (e.g. this == the instance of Weather class object).
    weather.on("data", function () {
    var payload = JSON.stringify({
      deviceId: deviceId,
      location: location,
      // celsius & fahrenheit are averages taken from both sensors on the shield
      celsius: this.celsius,
      fahrenheit: this.fahrenheit,
      relativeHumidity: this.relativeHumidity,
      pressure: this.pressure,
      feet: this.feet,
      meters: this.meters,
      lightLevel: this.lightLevel
    });

    // Create the message based on the payload JSON
    var message = new device.Message(payload);

    // For debugging purposes, write out the message payload to the console
```

```
    console.log("Getting data:" + message.getData());

    // Send the message to Azure IoT Hub
    client.sendEvent(message, printResultFor('send'));
    });
});

// Helper function to print results in the console
function printResultFor(op) {
return function printResult(err, res) {
    if (err) console.log(op + ' error: ' + err.toString());
};
}
```

4. Stream Analytics Query

Por último, se incluye el código utilizado en la consulta o query de Stream Analytics. El objetivo de este código es consolidar los datos que se reciben de la placa a cada segundo, de manera que aparezca la media horaria almacenada en la nube.

```
SELECT
   AVG(fahrenheit) TempF,
   AVG(celsius) TempC,
   AVG(relativeHumidity) Humidity,
   AVG(pressure) Pressure,
   AVG(meters) AltM,
   AVG(CAST(lightLevel as float)) Light,
   location,
   deviceId,
   sensorName,
   System.Timestamp AS Timestamp
INTO
   [WeatherBI]
FROM
   [WeatherShield]
GROUP BY
   TumblingWindow (second, 3600), deviceId, sensorName, location
```

5. Referencias

El código utilizado en este proyecto se ha obtenido de la página web citada a continuación, que se ha utilizado también como referencia a la hora de configurar la placa.

Node.js - Connected Weather Station. – thinglabs.io [Online]

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MODELO PREDICTIVO. MACHINE LEARNING APLICADO AL ANÁLISIS DE DATOS CLIMÁTICOS CAPTURADOS POR UNA PLACA SPARKFUN.

Anexo II

DIAGRAMA GENERAL

Autor: Diego Iribarren Baró

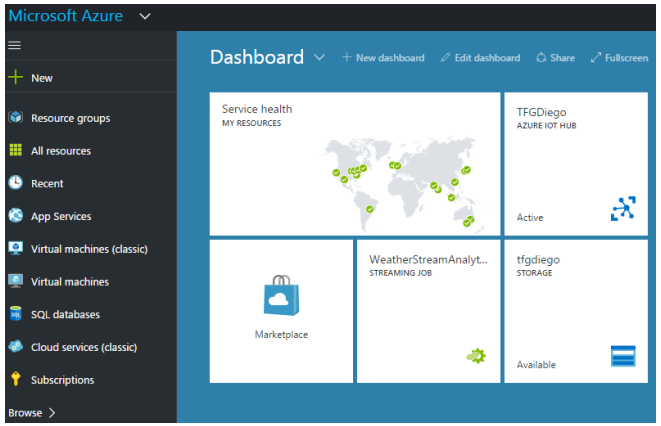
Director: Jaime Pereña Pinedo

PLACA SPARKFUN

Tareas realizadas:

Arduino	Preparación y configuración del dispositivo
Visual Studio	Desarrollo del código

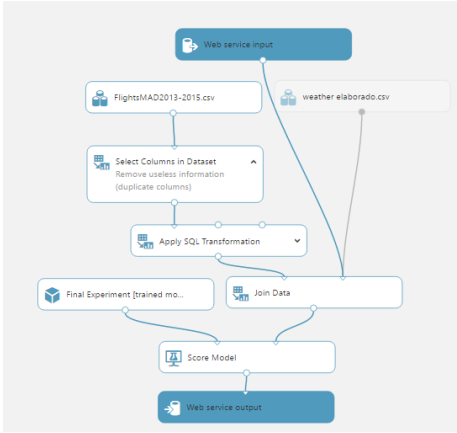
ENTORNO EN LA NUBE



Tareas realizadas:

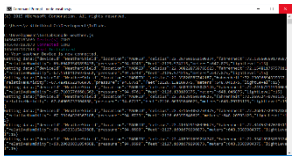
Azure	Configuración del entorno
Device Explorer	Conexión del dispositivo
Stream Analytics	Transmisión de datos Consolidación Almacenamiento Visualización

MODELO PREDICTIVO



Tareas realizadas:

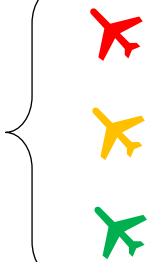
ML Studio	Creación del experimento
Excel ML Studio	Preparación de los datos Meteorológicos Vuelos
ML Studio	Elección del algoritmo Despliegue del modelo



AEMet
Agencia Estatal de Meteorología

Aena
Aeropuertos Españoles y Navegación Aérea

Machine Learning Studio



UNIVERSIDAD PONTIFICIA COMILLAS
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
GRADO EN INGENIERÍA ELECTROMECÁNICA