IMPLEMENTACION EN AZURE CLOUD DE UN SISTEMA SIMULADO DE EFICIENCIA ENERGETICA, PREVENCION DE FALLOS DE CALIDAD Y VIRTUALIZACION DE SENSORES PARA UNA INDUSTRIA DE LAMINACION DE TUBOS EN CALIENTE

Julen Manzano Lejarza
Grado de ingeniería informática
Inteligencia Artificial

Nombre Consultor: David Isern Alarcón
Nombre Profesor responsable: Carles Ventura Royo

Junio 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Implementación en Azure Cloud de un sistema de eficiencia energética, prevención de fallos de calidad y virtualización de sensores para una industria de laminación de tubos en caliente.*
Nombre del autor:	*Julen Manzano Lejarza*
Nombre del consultor/a:	*David Isern Alarcón*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega (mm/aaaa):	junio/2018
Titulación::	*Grado ingeniería informática*
Área del Trabajo Final:	*Inteligencia artificial*
Idioma del trabajo:	
Palabras clave	*Azure, Tiempo real, Machine Learning*

Resumen del Trabajo (máximo 250 palabras):

Los sistemas de inteligencia artificial dentro del sector industrial están siendo respaldados por la entrada de los servicios 'Serverless' en Cloud, facilitando no solo la computación distribuida hasta niveles de escalabilidad antes inalcanzables, sino el despliegue y desarrollo de proyectos que permiten la abstracción de los problemas asociados al despliegue, escalabilidad, mantenimiento o seguridad.

Mediante el uso de estas arquitecturas, en este caso Azure de Microsoft, logramos crear modelos analíticos avanzados para la determinación de sensores virtuales, detección de anomalías energéticas y supervisión de un sistema de calidad que son introducidos en un proceso de gestión de la información en tiempo real que nos permite, tras pasar por los modelos de comportamiento obtenidos, obtener una información de salida determinante para el correcto funcionamiento de los equipos industriales, y lo más importante, capaz de dotar a nuestra fábrica de información relevante en el mismo instante en el que ésta se produce, permitiendo, en una segunda fase, integrarse con un sistema de decisiones automatizado mediante el uso de la misma arquitectura de comunicación, actuando contra los equipos al modificar su configuración óptima en cada instante del proceso.

Esta información, es ofrecida en un panel de mandos o Business Intelligence que nos muestra no solo la información recogida y la procesada en tiempo real, sino a la vez la información histórica que permitirá al experto la toma de decisiones estratégicas basada en información y experiencia.

Abstract (in English, 250 words or less):

The artificial intelligence systems in the industrial sector are being supported by the entry of serverless services in the cloud, allowing distributed computing to a degree of scalability which was impossible before, as well as the deployment and development of proyects that allow the abstraction of issues related to deployment, scalability, maintenance and securiy.

By using these architectures, in this case Microsoft's Azure, we can create advanced analytic models for the designation of virtual sensors, detection of energetic anomalies and the overseeing of a qualified system that are put into a process of information management in real time that allows us, after going through the behaviour models obtained, to get an information output crucial for the proper operation of industrial systems, and what is more important, able to add relevant information to our factory in the same moment it is created, allowing its use by an automatized decision-making system using the same information architecture, acting on the systems optimizing their configuration at every moment.

This information is offered in a control panel or business intelligence that shows us not only the gathered and processed data in real time, but also a history of information that will allow the expert to take strategic decisions based on information and experience

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

La industria se encuentra inmersa en la entrada a la denominada cuarta revolución industrial *(Industry 4.0)* mediante la digitalización de las cadenas de valor a través de la integración del procesado de datos, logrando controlar, producir y predecir de forma más eficiente.

En una fase o estadio cero, podemos decir que la industria dispone de diversas fuentes de información provenientes de distintos departamentos en diferentes formatos, tales como bases de datos, aplicaciones propias o de terceros, PLCs, excel, videos, audios o incluso emails.

El análisis de información se realiza mediante la creación de informes que solo abarcan los propios sistemas de creación de datos (informes contables, de recursos humanos, etc), siendo estos independientes y rara vez capaces de fundir información de diversos departamentos, alcanzando un proceso éste, la veces que se alcanza, costoso y poco eficiente además de propenso a múltiples errores humanos.

El primer estadio se alcanza cuando se instaura un proceso de Extracción de información automatizada *(ETL)* de cada una de las fuentes de datos existentes en la empresa y las transforma y agrega hasta volcarlas en un DataWarehouse optimizado para la creación de distintos DataMarts que cumplirán funciones específicas de visualización o análisis básico de la información según el objetivo deseado.

Estos procesos se automatizan, disponiendo de información reciente sin necesidad de intervención humana, creada en lotes de tiempo establecidos para que la extracción de información no influya en la parte operacional de la industria.

Los cuadros de mando o Business Intelligence se centran y permiten visualizar datos históricos, compararlos y analizarlos con estados temporales anteriores, así como exportar los datos a otros sistemas de información paquetizada y agregada según las necesidades de cada usuario.

En este proyecto abarcamos los dos siguientes estadios, considerando que la industria se encuentra en disposición de acceder a ellos.

En el estadio segundo, que podríamos denominar ***Analítica de Datos Avanzada***, los datos recogidos y almacenados en DataMarts nos permite obtener distintos modelos de comportamiento así como información hasta ahora oculta en las fuentes de datos originales.

Estos modelos nos permiten por lo tanto incorporar nuevos valores, KPIs o parámetros que nos ayuden a medir, configurar y optimizar el rendimiento de un equipo o planta.

En este punto, somos capaces de mejorar nuestra toma de decisiones al incrementar la información obtenida. Podemos por ejemplo comparar cual es el valor de consumo respecto a un valor óptimo, y modificar las configuraciones para cotejar posibles mejoras o deficiencias en este o cualquier otro aspecto, si bien este proceso se realiza una vez se ha obtenido la información, es decir, corregimos después de producirse la acción, no antes.

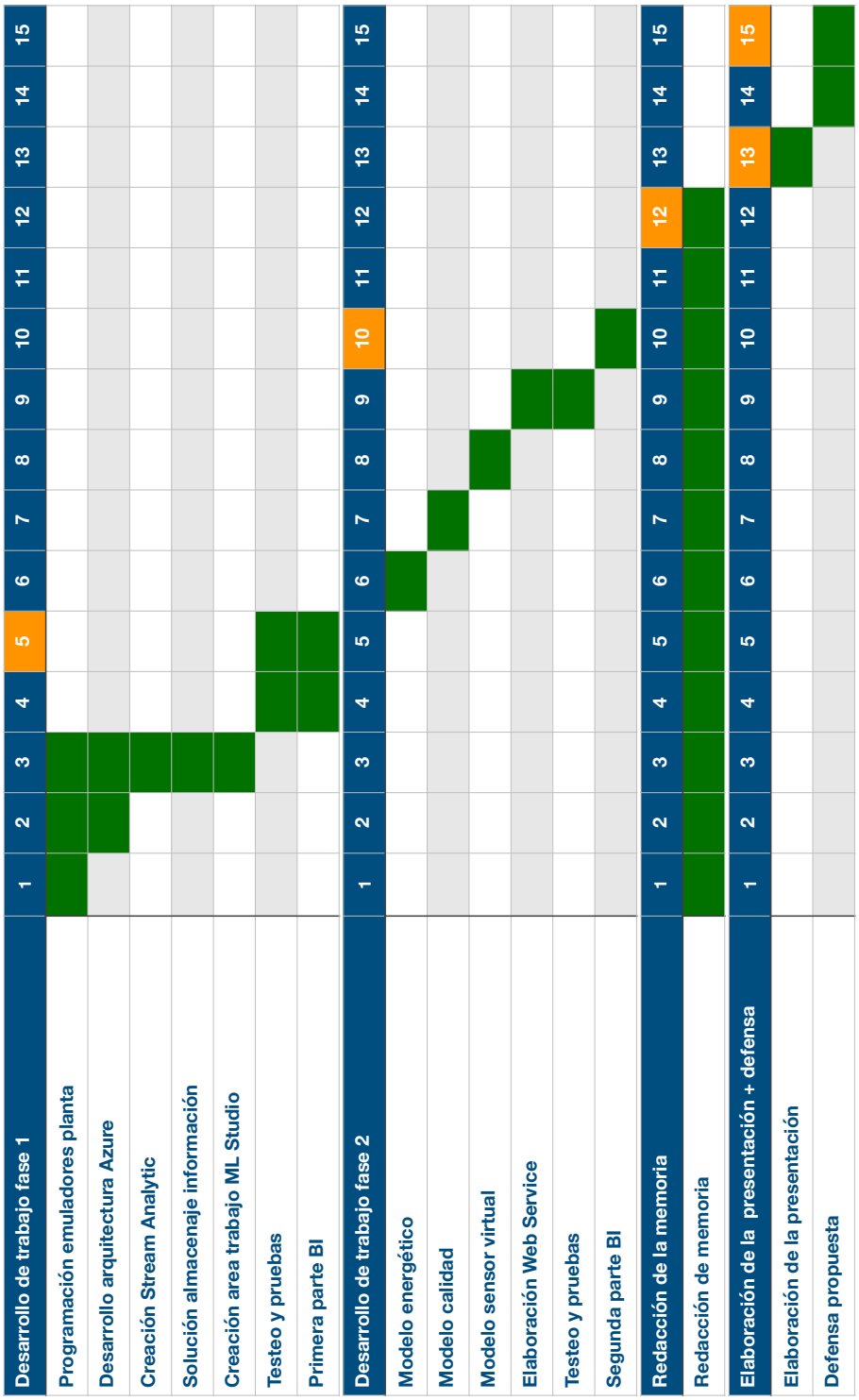
El siguiente estadio que trataremos, incorporamos el tiempo real. Si somos capaces de computar y procesar la información que obtenemos en una ventana temporal cercana a la producción de la información, de acuerdo a los modelos obtenidos seremos capaces de indicar cual es la configuración óptima para que el consumo real, por poner un ejemplo, se asemeje al consumo óptimo. En este punto podemos tanto corregir una acción para que no llegue a producirse un hecho, como fomentar las acciones necesarias para que un hecho se produzca.

1.2 Objetivos del Trabajo

1.2.1. Objetivos generales

. Diseño e implementación de una arquitectura en Azure Cloud que permita procesar los datos en diversos flujos de información para lograr visualizar en tiempo real posibles comportamientos energéticos anómalos, la probabilidad de que la pieza creada no alcance los parámetros de calidad establecidos y el cálculo del parámetro de una piscina de tratamiento de tubos mediante la virtualización de un

sensor, todo ello para un equipo de producción de laminado



en caliente de tubos.

El entorno de fabricación no es relevante para el desarrollo del presente trabajo fin de grado, y simplemente nos sirve para, en ocasiones, poder contextualizar la información en un entorno de trabajo real.

La fuente de datos pertenece a distintos entornos y fuentes de producción y nos permitirá desarrollar distintas técnicas de machine y deep learning hasta alcanzar los modelos óptimos para cada caso de uso concreto.

1.2.2. Objetivos específicos

. Creación de un emulador de sensores en JAVA utilizando las librerías de conexión de Azure, el código necesario para emular distintos sensores que emitan con un delay determinado una serie de datos de planta que nos permitan emular el funcionamiento de la misma.

. Creación de distintos modelos de analítica de datos avanzada englobados dentro de los métodos supervisados y que traten tanto problemas de clasificación como problemas de regresión.

. Creación de los flujos de datos desde la obtención de los valores, su paso por los web service de los modelos para la obtención de los valores calculados hasta su posterior visualización en PowerBI como cuadro de mandos así como su almacenamiento en un Blog Storage, desde donde volveremos a entrenar nuestros modelos para que sean optimizados periódicamente.

1.3 Planificación del Trabajo

1.4 Breve sumario de productos obtenidos

. Memoria del trabajo
. Código fuente y Jar ejecutable del emulador de los sensores de planta
. Vídeo del sistema en funcionamiento
. Presentación del proyecto
. Implementación en Cloud funcional

1.5 Breve descripción de los otros capítulos de la memoria

A lo largo del segundo capítulo de la memoria nos centramos en el diseño e implementación de la arquitectura serverless en cloud de Azure que nos dará servicio al proyecto. Desarrollamos cada uno de los servicios necesarios para el correcto funcionamiento del sistema, así como el desarrollo del código que nos permita emular dispositivos con diversos sensores mediante el uso de JAVA.

Tras su testeo y comprobación utilizando un número aleatorio de sensores desarrollaremos los análisis de datos necesarios para determinar un modelo de calidad, energético y de sensor virtual que será tratado en el tercer capítulo del presente documento.

A lo largo del cuarto capítulo modificaremos y adaptaremos la arquitectura desarrollada anteriormente para dar capacidad de trabajo en tiempo real a las variables ya definidas en el capítulo tercero. De esta forma dispondremos de un sistema completo de análisis en tiempo real.

El quinto capítulo nos permitirá desarrollar un cuadro de mandos donde mostrar tanto la información recibida en tiempo real como la información histórica transformada para dotar de inteligencia al negocio.

En el sexto capitulo abordaremos las posibilidades de ampliación del sistema dentro del entorno de la inteligencia artificial.

2. Creación de la arquitectura básica

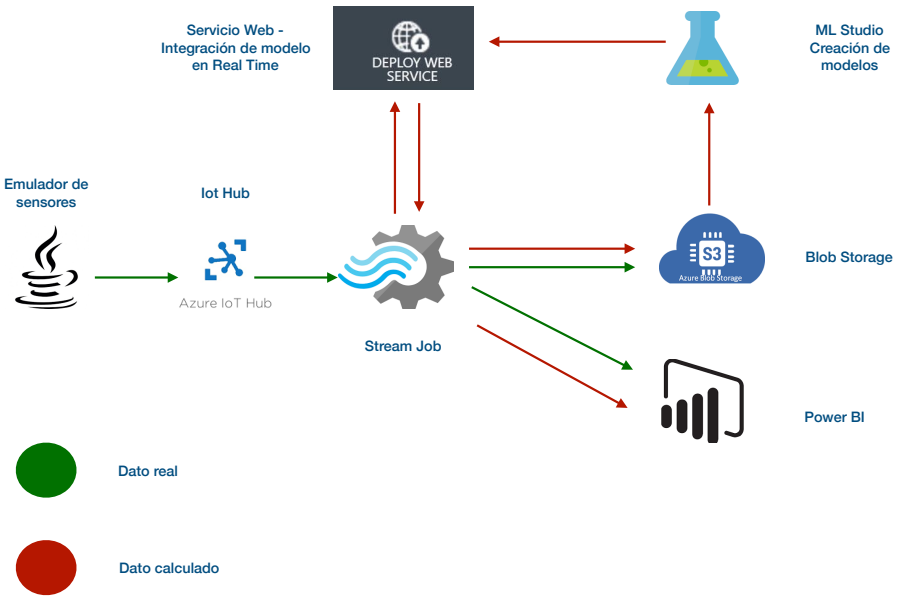
2.1 Resumen y descripción básica

Es necesario definir y crear una arquitectura de servicios Cloud previa que podamos ir modificando y adaptando de acuerdo a la información que procesemos. La información de entrada será emulada mediante la programación de código en JAVA utilizando las librerías de AZURE que permiten dotar de funciones de conectividad hacia su nube de forma bidireccional con los dispositivos existentes en planta.

El almacenaje de la información sin procesar que se recibe del emulador se realizará en un contenedor de Blobs, y será visualizada simultáneamente en un cuadro de mandos ejecutado con **PowerBI.**

Como nuestro objetivo va mas allá de realizar un almacenaje y una visualización de la información, utilizaremos los datos contenidos y almacenados en ficheros externos para realizar los procesos de analítica avanzada de datos que nos permita, mediante el uso de técnicas tanto de machine como de deep learning, la creación de modelos de comportamiento donde obtener parámetros de optimización de calidad y de consumo energético, así como el desarrollo de un sistema de sensorización virtual.

Estos modelos, creados en cloud, en **MLStudio**, servirán para crear un WEB Service que absorberá los datos de entrada, los cuales serán procesados bajo el modelo de comportamiento creado, para ofrecer en tiempo real la información de salida ya procesada, de acuerdo al modelo óptimo calculado, siendo esta información almacenada en nuestro Blob Storage, con el objetivo final de ser utilizados recurrentemente en el análisis de datos, permitiendo una mejora continua de los modelos.



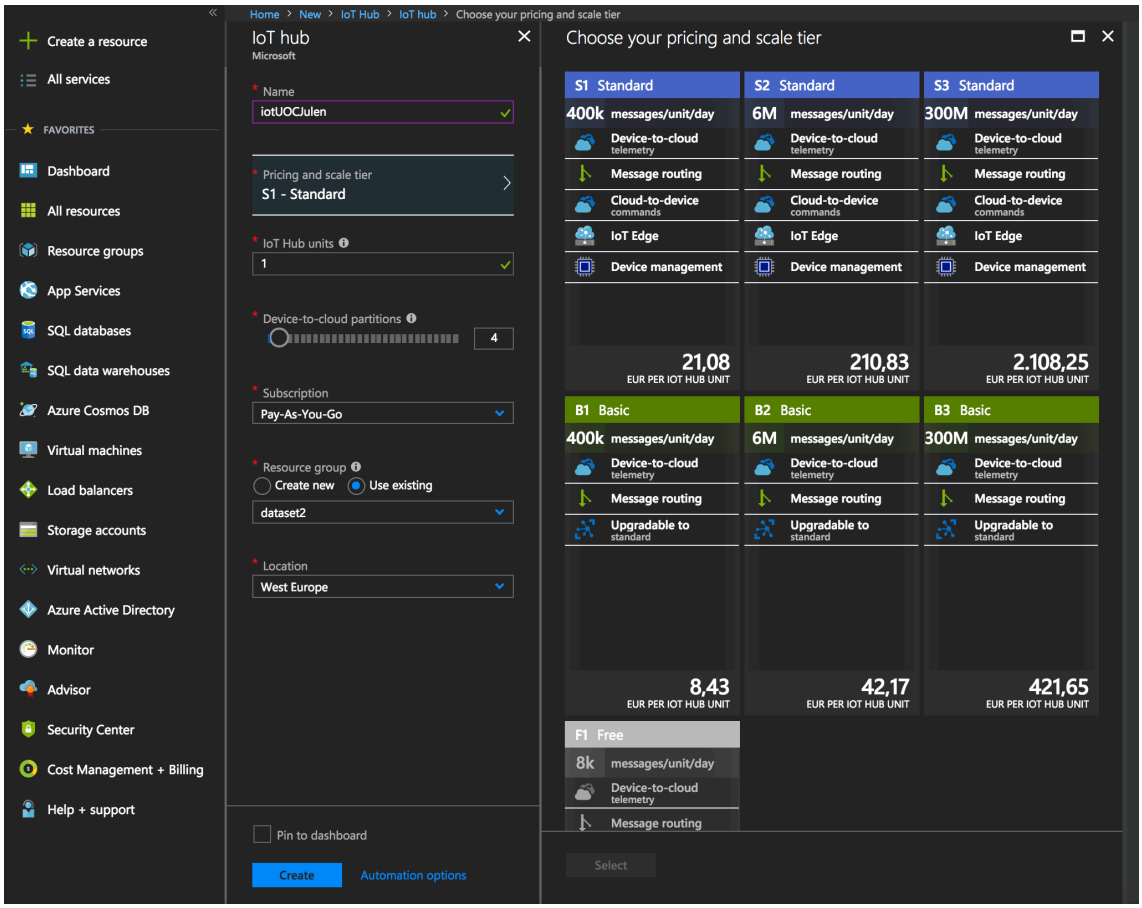
(Figura 2: Arquitectura básica Azure Cloud)

2.2. Creación de los servicios

2.2.1 IotHub

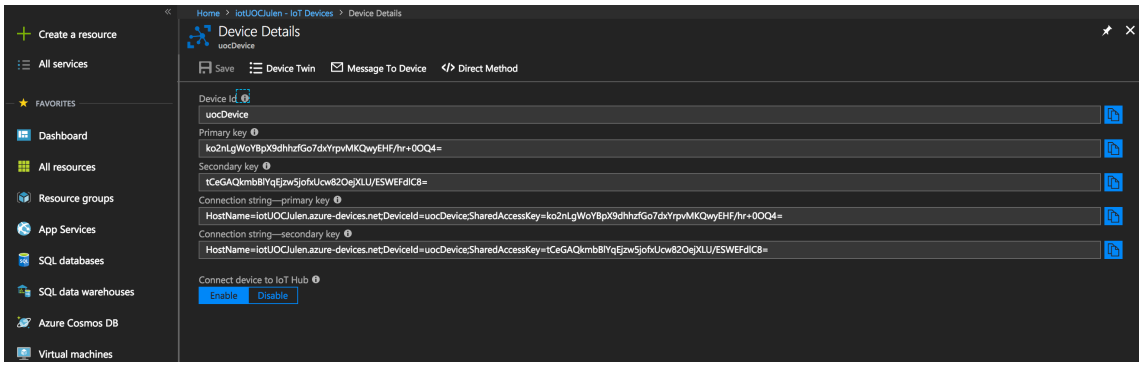
El IotHub es un servicio de Azure que permite conexiones bidireccionales con múltiples dispositivos. Este concentrador de dispositivos se ofrece en diversas soluciones y costes basadas en el número de mensajes que son capaces de recibir diariamente.

En nuestro caso, hemos seleccionado un **IoT Hub S1 Standar,** con capacidad para 400 K mensajes diarios y la incluiremos dentro de un grupo de recursos existentes que nos permitirá agrupar los servicios por proyecto.



(Figura 3: Creación de un IoT Hub)

Dentro de los IotHub creamos el dispositivo que nos identificará en el Cloud cada uno de los sensores o elementos emisores de información. Al crear un nuevo dispositivo disponemos de un identificador y dos claves, primaria y secundaria (usada ante problemas de comunicación con la clave primaria), con un String de conexión que asociado a cada paquete de información nos ayudará a conectar los dispositivos emisores con nuestro servicio en Azure.

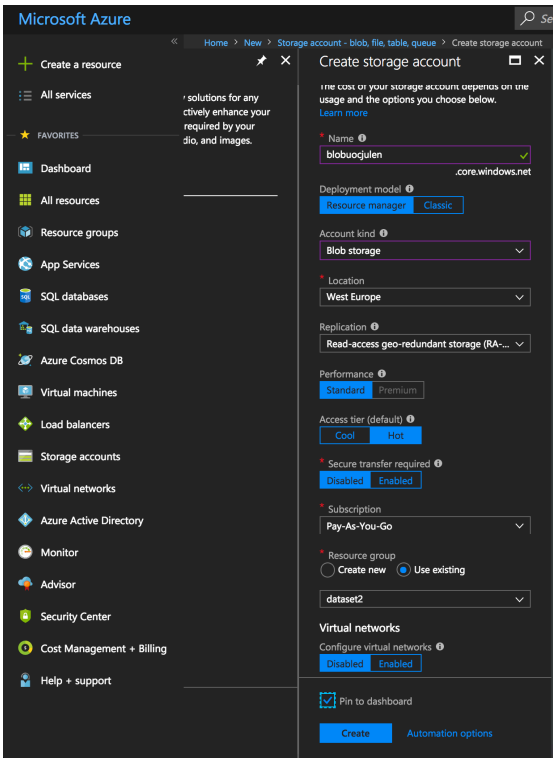


(Figura 4: Cadenas de conexión de un dispositivo en Iot Hub)

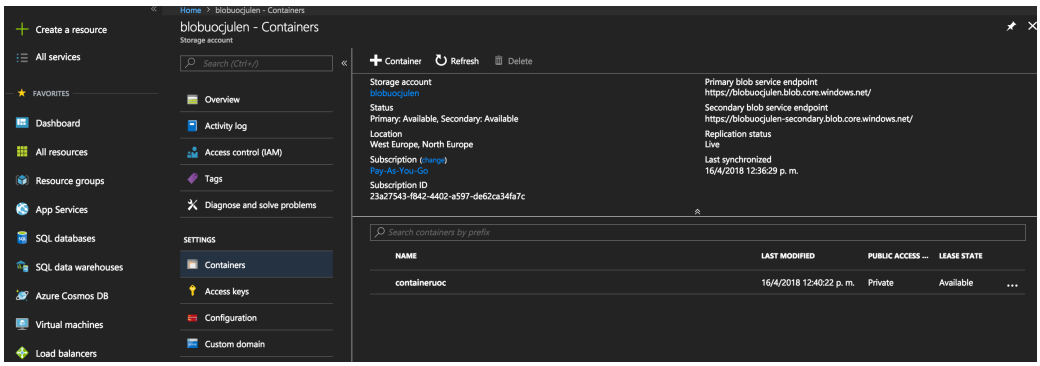
2.2.2 Blob Storage

Este servicio nos capacita para almacenar objetos de forma masiva sin preocuparnos por su escalabilidad y disponibilidad, y se encuentra especialmente orientado al almacenamiento de datos no estructurados.

Determinado el identificador del servicio, el cual será accesible mediante mqtt, su ubicación, el grupo de recursos al que lo asociaremos, la política de replicación y de seguridad de acceso creamos un contenedor nuevo dentro de nuestro Blob, que será el espacio donde almacenaremos los datos de entrada y posteriormente los datos calculados.



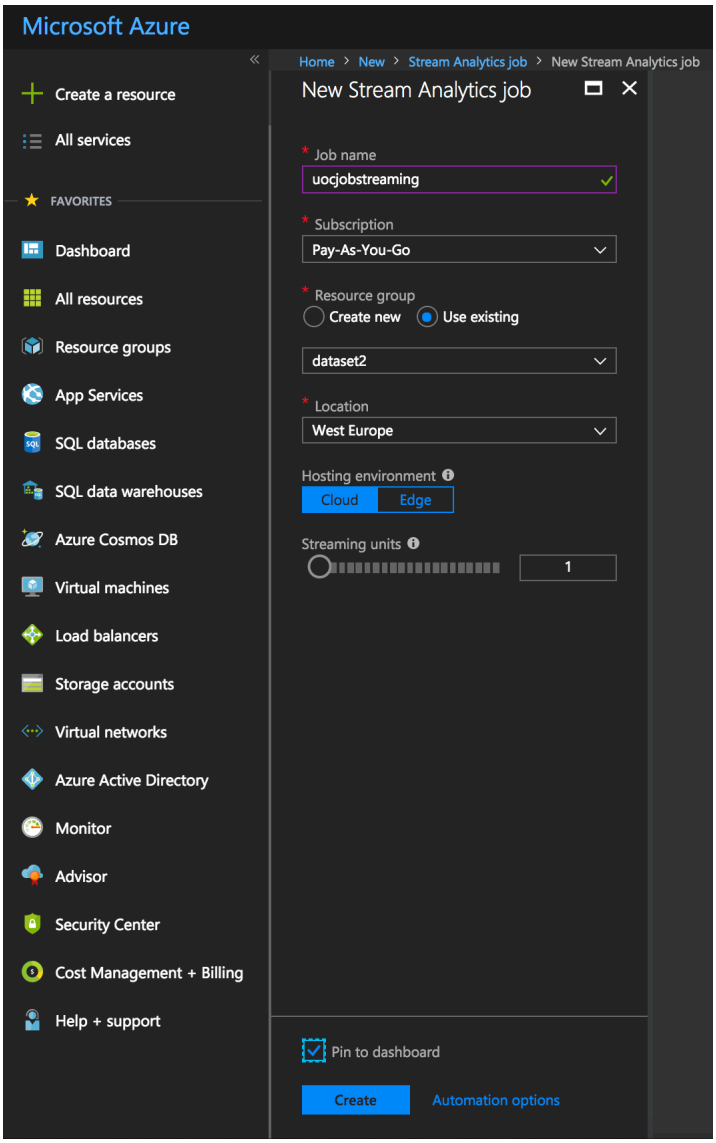
(Figura 5: Creación de un Blob Storage)



(Figura 6: Creación de un contenedor)

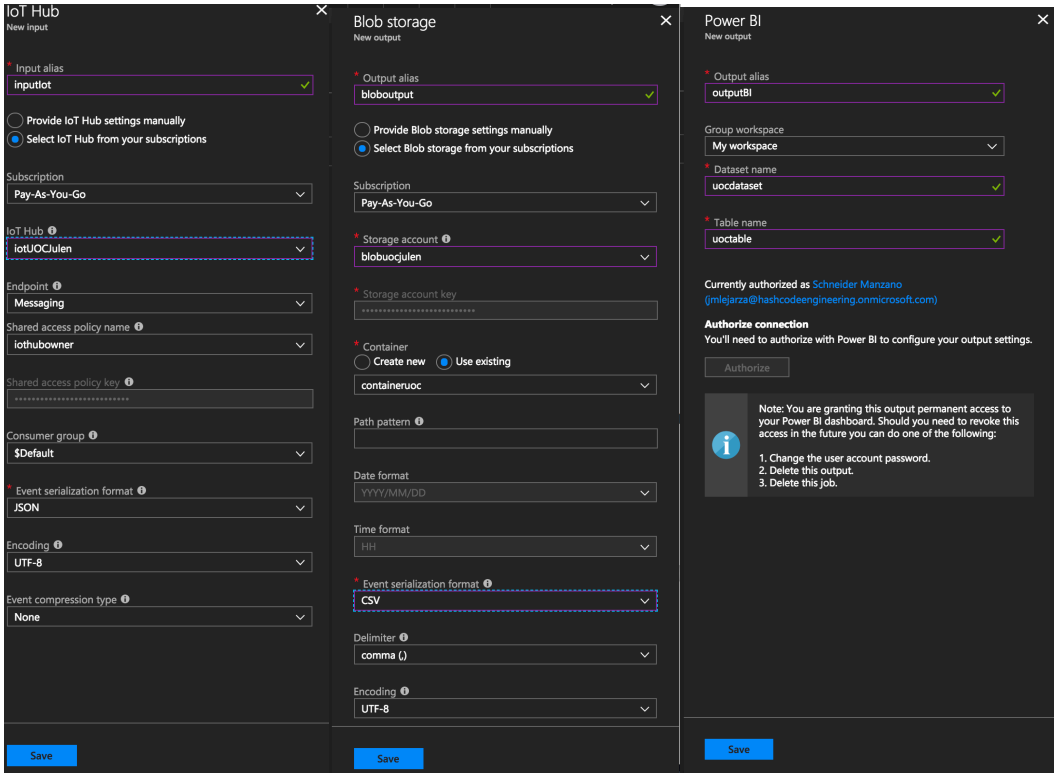
2.2.3 Job Streaming Analytics

Hasta este momento disponemos de un punto de entrada y un punto de almacenaje de la información, pero requerimos de la definición de un canal que comunique ambos espacios. Es en este punto donde debemos definir nuestro Job Streaming.



(Figura 7: Creación de un Stream Analytics Job)

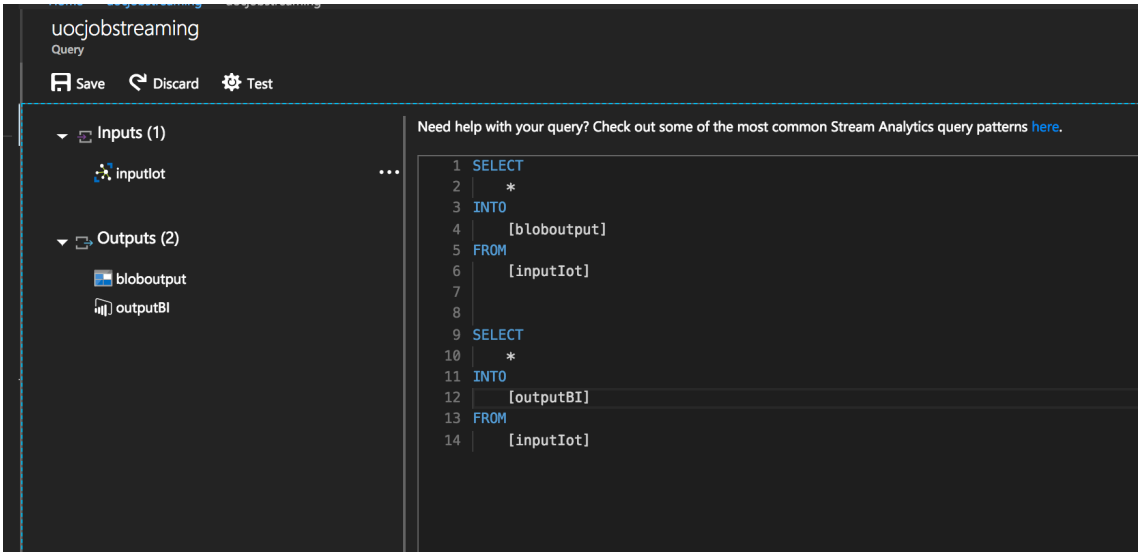
Una vez creado el canal podemos comenzar a definir los puntos de entrada y de salida y mediante querys en lenguaje SQL especificar los nexos de unión entre ellos.



(Figura 8: Determinación de input desde IotHub y dos output al Blob y al cuadro de mandos)

En este punto hemos definido un punto de entrada de la información a partir del IotHub anteriormente definido, el punto de salida será nuestro contenedor Blob creado con anterioridad, así como una salida al visualizador PowerBI

Las querys nos permitirán unir ambos extremos para terminar de definir el flujo de datos.

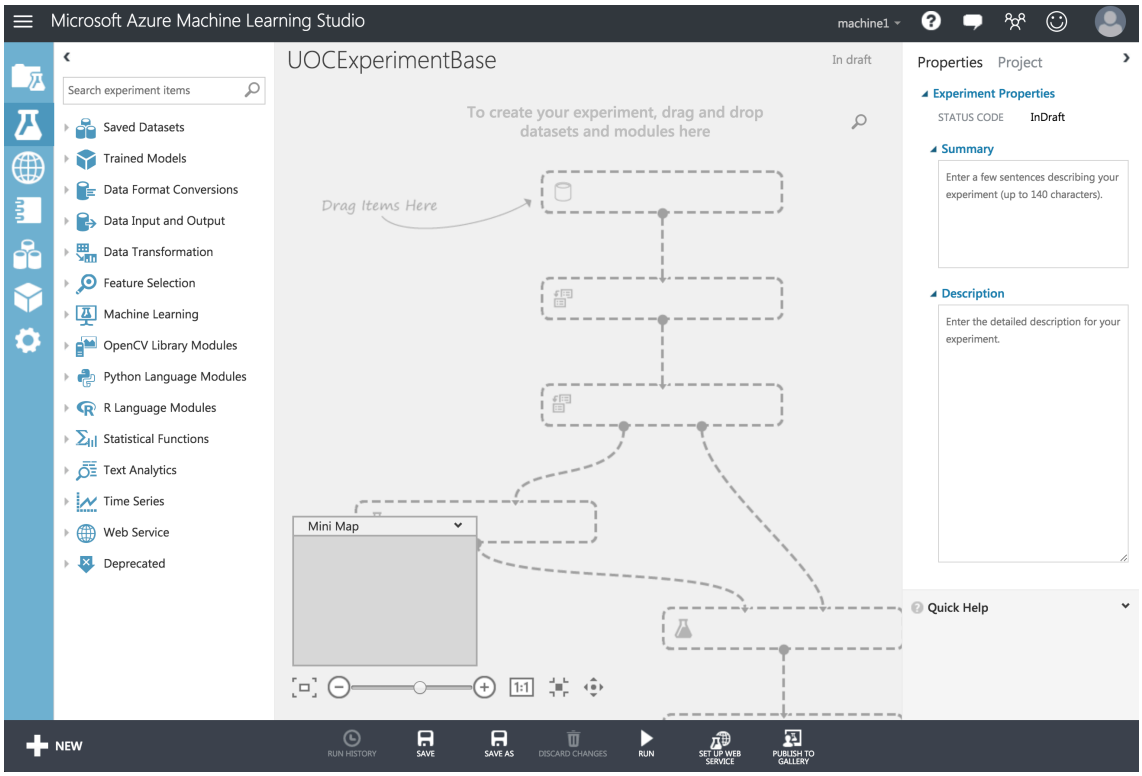


(Figura 9: Determinación de la query)

2.2.4. ML Studio

ML Studio es un entorno que nos permite realizar análisis complejos de analítica de datos avanzada y a la vez crear servicios web sencillos basados en funciones y plenamente integrados con el entorno Serverless de Azure. Si bien la mayoría de operaciones de limpieza, transformación y análisis deben realizarse mediante la creación de código ad hoc en Python o R, nos permite simplificar pequeñas labores de captación de datos o determinación de modelos.

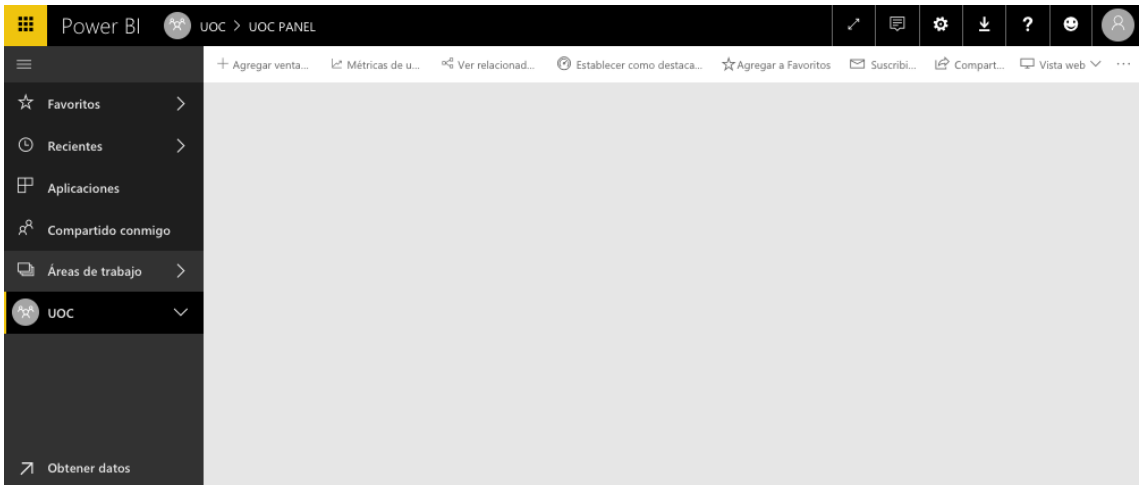
El verdadero potencial de ML Studio radica en su capacidad de integración con la plataforma Cloud, pudiendo crear Web Service que bajo el nombre de una función determinada se integrará en nuestro Stream Job bajo un nuevo canal de transmisión de información que nos devolverá valores calculados según el modelo de comportamiento generado.



(Figura 10: Panel ML Studio)

2.2.5. PowerBI

Un potente visualizador integrado dentro de las soluciones de Azure, capaz de tratar tanto información en tiempo real como recoger la información de los Blob Storage donde la hemos almacenado y realizar un proceso de Extracción, Transformación y Carga con el que visualizar información histórica en un mismo entorno.



(Figura 11: Panel Power BI)

2.3. Código emulador de sensores

Para simular un dispositivo que emite información procedente de distintos dispositivos creamos un proyecto de Maven en JAVA donde agregaremos las siguientes dependencias

```
<dependency>
  <groupId>com.microsoft.azure.sdk.iot</groupId>
  <artifactId>iot-device-client</artifactId>
  <version>1.3.32</version>
</dependency>
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.3.1</version>
</dependency>
```

Importamos las siguientes librerías a nuestro proyecto

```
import com.microsoft.azure.sdk.iot.device.*;
import com.google.gson.Gson;

import java.io.*;
import java.net.URISyntaxException;
import java.util.Random;
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;
```

Se adjunta copia del código principal en el anexo I, así como copia del proyecto completo junto a la presente memoria.

Nuestro código emulará con el objetivo de testar la arquitectura un dispositivo con cinco sensores que emitirán valores contra nuestro IotHub de forma aleatoria.

(Figura 12: Código principal emulador JAVA)

```
    public static void main( String[] args ) throws IOException, URISyntaxException {
          //connString = (args [0]).toString();
          client = new DeviceClient(connString, protocol);
          client.open();
          //System.out.println(connString);
          MessageSender sender = new MessageSender();

          ExecutorService executor = Executors.newFixedThreadPool(1);
          executor.execute(sender);

          System.out.println("Press ENTER to exit.");
          System.in.read();
          executor.shutdownNow();
          client.closeNow();
        }

}

                    int int_sensor4  = (int) (Math.random()*21);
                    int int_sensor5  = (int) (Math.random()*29);

                    System.out.println("Número de mensaje enviado "+ contador_envios);
                    contador_envios ++ ;

                    TelemetryDataPoint telemetryDataPoint = new TelemetryDataPoint();

                    telemetryDataPoint.testdevice_sensor1 = int_sensor1;
                    telemetryDataPoint.testdevice_sensor2 = int_sensor2;
                    telemetryDataPoint.testdevice_sensor3 = int_sensor3;
                    telemetryDataPoint.testdevice_sensor4 = int_sensor4;
                    telemetryDataPoint.testdevice_sensor5 = int_sensor5;

                    // Serializamos el objeto para crear un mensaje que contenga la información
                String msgStr = telemetryDataPoint.serialize();
                Message msg = new Message(msgStr);
                msg.setMessageId(java.util.UUID.randomUUID().toString());
                System.out.println("Sending: " + msgStr);

                // Creamos una respuesta del IotHub que nos permita identificar el estado de la recepción de los mensajes
                Object lockobj = new Object();
                EventCallback callback = new EventCallback();
                client.sendEventAsync(msg, callback, lockobj);

                synchronized (lockobj) {
                  lockobj.wait();
                }

                // Tiempo determinado entre envíos en mlsg
                Thread.sleep(2000);
              }
            } catch (InterruptedException e) {
              System.out.println("Finished.");
            }
          }
        }

                context.notify();
              }
            }
          }
        }
```

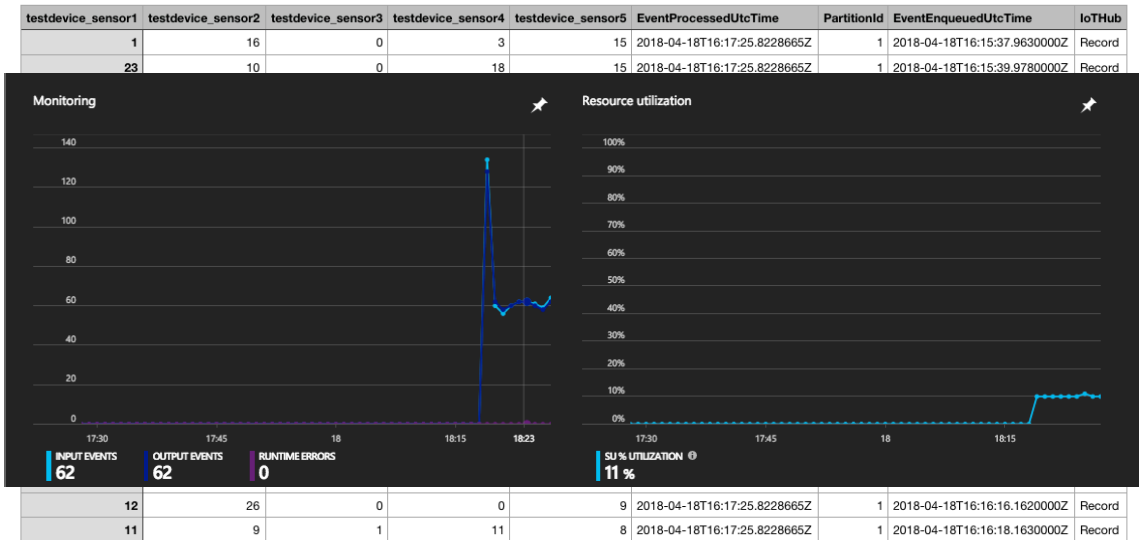
2.4. Test arquitectura

Finalizado el diseño y la implementación de la arquitectura que nos servirá como base de nuestro sistema de IA, es el momento de testarla poniendo en funcionamiento el código JAVA emulador de dispositivo y observando el tráfico de información en el cloud.

En el Stream Job observamos transcurrido un tiempo desde el inicio del emulador el número de mensajes de entrada y de salida que recibe el sistema.

(Figura 13: Panel de monitorización Stream Job)

En el Blob Storage podemos observar la creación de un fichero csv que contendrá los datos emulados, junto a un timestamp automático que nos indica por un lado la

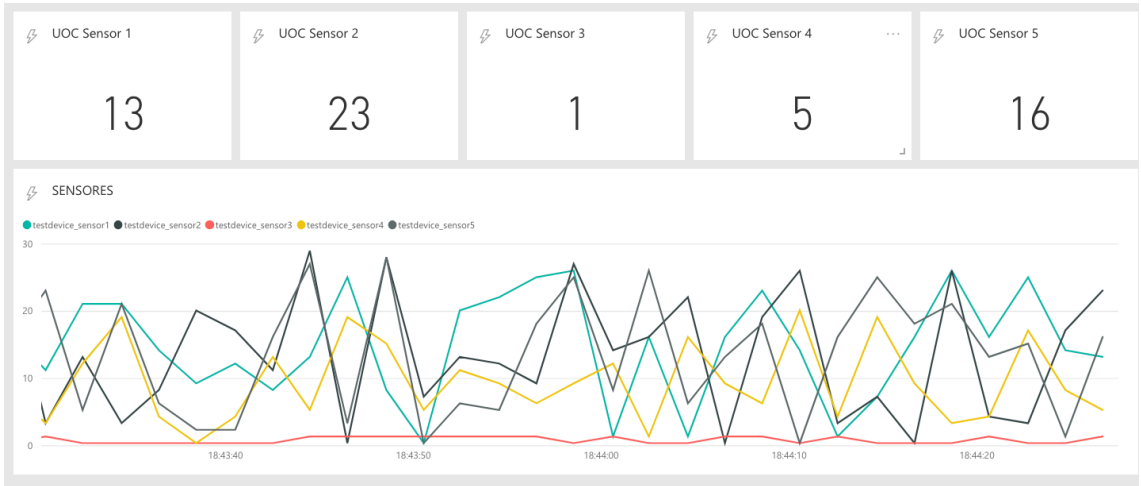


recepción del mensaje y por otro el momento de procesado por el sistema.

(Figura 14: CSV almacenado en el Blob)

Figura 15: Panel Power BI test)

Por último, desde PowerBI podemos disponer ya de los datos en tiempo real, los cuales se muestran en una breve aplicación de la siguiente forma.



3. Creación de modelos analíticos

3.1. Modelo de sensor virtual

3.1.1. Sentido de un sensor virtual

Todo proceso que requiera ser monitorizado debe disponer de uno o varios sensores que proporcionen información sobre dicho proceso. Según el tamaño del proceso y la información que deseemos recabar sobre el mismo requeriremos de un número de sensores proporcional a nuestras necesidades, pudiendo alcanzarse un número nada despreciable de equipos de captación de información.

Estos equipos de sensorización, en función de su grado de exactitud o del ambiente en el que se encuentran desplegados, pueden suponer un coste elevado además de requerir una tasa de reposición alta, o incluso, como es nuestro caso de estudio, tener que tomarse muestras para ser analizados en laboratorio mediante la aplicación de diversos procesos químicos y entregar un resultado sobre un parámetro decisivo para nuestra producción.

Ante esta situación, los sensores virtuales, también llamados sensores de software, observadores o estimadores de estado nos permiten disponer de la información requerida en un punto determinado del proceso sin necesidad de disponer del equipo físico.

El cálculo del valor de dichos sensores virtuales se realiza mediante técnicas de machine o deep learning, en donde, habiendo obtenido información de procesos similares, donde disponemos de los parámetros de entrada del proceso, y el valor del sensor real en ese punto temporal, podemos crear un modelo de comportamiento que emule la salida del sensor ante la entrada de los distintos parámetros que afectan al modelo.

Mediante esta tecnología podemos reducir notablemente los costes de monitorización y control de procesos.

3.1.2 Datos de entrada

Disponemos de la información de un proceso de planta que es medido cada hora, adquiriendo los siguientes datos:

- DeviceId: Identificador del grupo de sensores.

- CE: Valor de determinadas partículas existentes en el líquido. Dicho valor requiere de una medición realizada en laboratorio, por lo que su cálculo resulta complejo y costoso. No nos identifican el tipo de partícula o unidades de medida.

- Temp_CE: Temperatura del líquido en la barrica.

- PH: Valor del PH de la barrica.

- Turbidez: Valor de la turbidez de la barrica.

- Voltaje: Voltaje aplicado sobre la barrica.

- DisPlanta: Identificador de la planta.

- EventTime: Momento temporal de la medición

El Dataset se compone de 2377 filas de información. Analizamos cada columna de datos independientemente obteniendo los siguientes resultados, respecto a valores tales como el tipo de dato, su valor medio, mediana, rango de valores máximo y mínimo, desviación estandar, número de valores únicos y cantidad de valores no informados.

Este primer análisis es importante realizarlo de forma exhaustiva, puesto que la limpieza y la calidad del dato a procesar es vital para la obtención de modelos funcionales.

	Tipo	Media	Mediana	Valor mínimo	Valor Máximo	Desviación estándar	Valores únicos	Valores no informados
Devide	String						1	0
CE	int	14*10^7	2190	0	83*10^8	10*10^8	159	0
Temp_CE	float	23,98	23,29	18,54	45,32	2,82	119	0

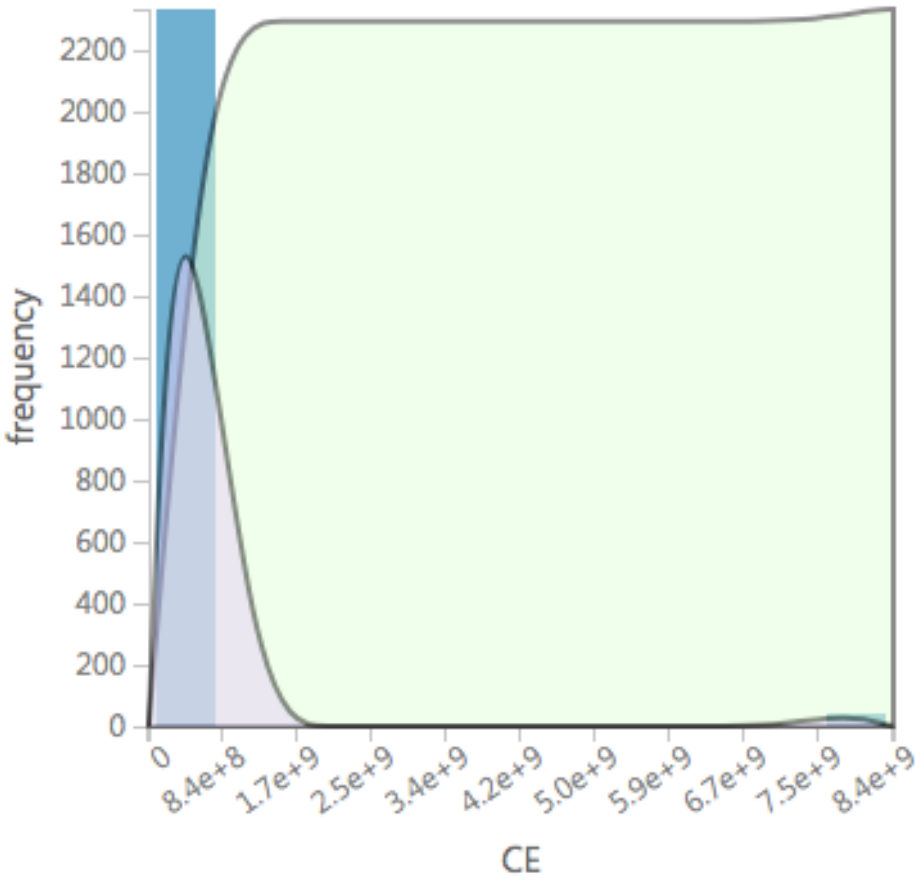
	Tipo	Media	Mediana	Valor mínimo	Valor Máximo	Desviación estándar	Valores únicos	Valores no informados
PH	float	6,8839	6,84	5,34	7,75	0,2983	71	0
Temp_PH	float	23,16	23,05	12,9	28,93	1,8575	109	0
Turbidez	float	253,267	222,52	4,2	1119,24	207,02	2325	0
Voltaje	float	490,80	489,38	5,13	600	25,75	90	0
DisPlanta	String						1	0
eventTime	String						2375	0

Figura 16: Tabla de características de datos de entrada

Tanto DeviceId como DisPlant observamos que son string que contiene identificadores con similar valor en la totalidad del Dataset, por lo que no aportan información relevante para el modelo y debemos eliminarlos.

Por otro lado, eventTime nos informa de que existen dos filas repetidas, puesto que disponemos de 2375 valores únicos, cuando tenemos en realidad 2377 filas de datos. Por ello, y para afinar aun mas el modelo debemos de localizar y eliminar esas dos filas redundantes.

Un aspecto importante a tener en cuenta, y que destaca respecto al resto de valores es la distribución de CE, que casualmente es nuestro valor objetivo.



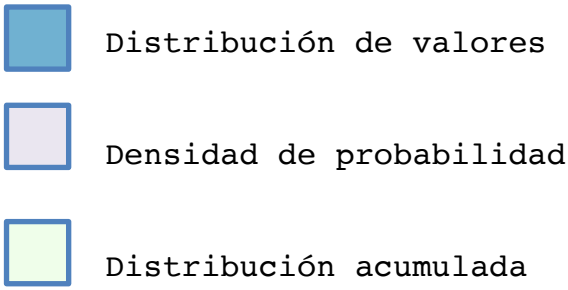


Figura 17: Distribución de variable CE

La distribución presenta claramente Outlyers que a priori, teniendo en cuenta que el proceso de inserción del dato se realiza desde un laboratorio, y probablemente no se encuentre automatizado, pudiera deberse a problemas de transcripción. Es vital la corrección del presente valor, ya que nos pervertirá totalmente el modelo. Por ello, tomamos como decisión anular las filas que contengan outlyers, si bien hubiéramos podido proceder con actuaciones tales como sustituir por la mediana o la media entre otras, o incluso crear un modelo que calcule los valores de esas dos filas para posteriormente incorporarlas al propio modelo de entrenamiento.

3.1.3 Modelos de regresión utilizados

Nos enfrentamos al estudio y creación de un modelo de regresión, en donde, a partir de valores numéricos obtengamos también un valor de carácter numérico.

Los modelos de regresión se basan en la relación de una variable que denominamos dependiente, respecto de una o un grupo de variables dependientes. A partir de esta estimación se obtiene lo que se conoce como una función de regresión, que nos permitirá calcular valores de salida a través de la entrada de los parámetros dependientes.

Los modelos utilizados para el entrenamiento y evaluación de los datos son:

- **Linear regression:** Este modelo establece una relación lineal entre las variables independientes, también llamadas explicativas o regresores y una variable de salida o variable dependiente.

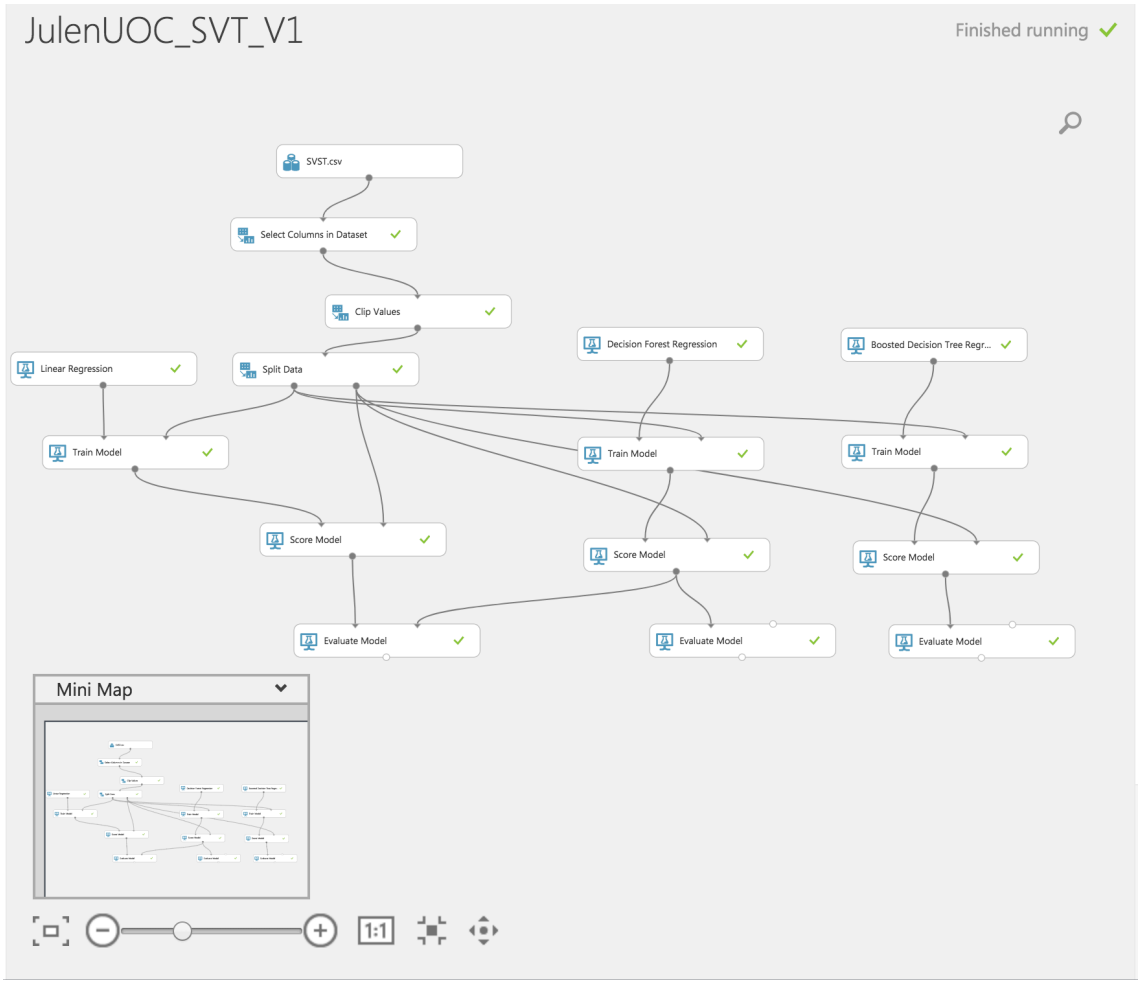


Figura 18: Panel ML Studio Sensor Virtual

La regresión lineal es un método estadístico utilizado en modelos simples que requieran de una tarea predictiva sencilla. Aunque este modelo pueda ser catalogado como simple, se han realizado evoluciones que permiten su uso y afinamiento en modelos, esos si, de baja complejidad.

El problema clásico de regresión trata una sola variable de entrada para obtener una única variable de salida, si bien en este caso concreto nos enfrentamos a múltiples variables de entrada que serán empleadas para obtener un único valor objetivo, por los que en esta situación hablamos de una regresión lineal multivariable.

Los métodos que MLStudio permite para ajustar la regresión consisten en el método de mínimos cuadrados ordinarios y el método de descenso de gradiente.

El método de mínimos cuadrados calcula el error como la suma del cuadrado de la distancia desde el valor real a la

línea de predicción y ajusta el modelo minimizando el error existente.

El descenso de gradiente minimiza la cantidad de errores en cada paso del sistema, permitiendo el establecimiento de distintos valores como pueden ser la velocidad de aprendizaje o el tamaño del paso.

Utilizo particularmente el método de mínimos cuadrados configurando la regresión lineal con regularización de pesos para evitar un overfitting que pudiera desvirtuar el modelo y asignando una semilla específica para obtener similares resultados ante similares entrenamientos.

El overfitting o sobreajuste se produce al entrenar un modelo en exceso provocando que el algoritmo sea altamente eficiente para los datos de entrenamiento pero se encuentre totalmente desajustado para los datos de test o datos reales. Es decir, el algoritmo se encuentra excesivamente ajustado ante unas situaciones de entrada muy particulares y no es efectivo ante nuevas muestras.

- **Decision Forest Regression:** Mediante este modelo alcanzamos una regresión lineal a través de el ensemble de árboles de decisión. Este tipo de árboles realizan una serie de pruebas simples que atraviesan los nodos de los árboles hasta alcanzar una hoja o decisión, permitiendo un coste computacional tanto en predicción como entrenamiento bajo y eliminando ruido no deseado en el modelo.

Se utiliza un bosque de decisión que genera una distribución gaussiana que combina la predicción de todos los árboles utilizados.

La configuración del modelo nos permite seleccionar diversos parámetros, entre los que se encuentra:

. *Método de ensambling*, pudiendo seleccionar el Bagging o agregación de boostrap, donde cada árbol es un bosque de decisión de regresión que genera una distribución gaussiana a modo de predicción, caso utilizado en nuestro modelo, o de replicación, donde cada árbol se entrena exactamente con los mismos datos de entrada.

. *Parámetros de entrada*, en donde podemos utilizar parámetros específicos si conocemos aquellos argumentos que por experimentación son mas efectivos para la creación del modelo, o bien por permitir que el modelo se entrene con

diversas muestras de parámetros hasta alcanzar el mas eficiente, opción seleccionada en nuestro caso.

. *Número de árboles de decisión:* Al crear más árboles mejoramos el modelo a costa de un mayor requerimiento de computación.

. *Profundidad máxima de los árboles de decisión:* Nos permite limitar su profundidad, pudiendo aumentar el ajuste a costa de correr el peligro de alcanzar un sobre ajuste del modelo y un incremento del tiempo computacional de entrenamiento.

. *Número de divisiones aleatorias por nodo:* Nos muestra la cantidad de divisiones por cada nodo utilizado.

. *Número mínimo de mientras por nodo:* Determinamos el número mínimo de valores requeridos para crear una nueva hoja

Al usar un rango de parámetros permitimos que el modelo interactúe hasta encontrar el ajuste de parámetros que nos ofrece un modelo mas eficiente, todo ello a costa de un coste computacional mas elevado. Al disponer de pocas filas de datos podemos permitirnos este sistema, que no sería eficiente en modelos de entorno reales que pueden alcanzar fácilmente los millones de filas de información.

- **Boosted Decision Tree Regression**

Este modelo nos permite crear múltiples árboles de decisión que son dependientes de árboles anteriores, aprendiendo automáticamente el algoritmo en base al ajuste de los árboles que le precedieron.

Del mismo modo que en el modelo utilizado anteriormente, este modelo nos permite seleccionar un rango de parámetros que nos ayude a encontrar la configuración óptima de forma eficiente.

Los parámetros de configuración en este caso son:

. *Número máximo de hojas por árbol:* Cantidad máxima de hojas que puede crear un árbol. Debemos tener especial cuando de no incrementar el valor excesivamente para no encontrarnos con un sobre ajuste no deseado.

. *Número mínimo de muestras por nodo:* Número requerido de valores que caen en el nodo requeridas para crear una nueva hoja en el árbol.

. *Tasa de aprendizaje:* Determina a que velocidad se converge a una solución óptima.

. *Número de árboles construidos:* Identificamos el número máximo de arboles que pueden ser construidos, incrementando la optimización del modelo a coste de una mayor necesidad computacional y de un riesgo de sobreajuste.

. *Número de semilla:* Garantiza los mismos resultados para similares valores de entrada.

3.1.4 Evaluación de los modelos

Los modelos de regresión devuelven una serie de métricas que nos permiten conocer la eficiencia del modelo calculado en base a una estimación del error entre los valores observados y calculados utilizados para realizar el test.

. **MAE (Error absoluto medio):** Mide la distancia entre las predicciones y los resultados reales. Un valor bajo indica mayor cercanía entre los valores.

. **RMSE (Error cuadrático medio):** Crea un valor capaz de resumir el error del modelo, que al utilizar los cuadrados elimina los signos del error.

. **RAE (Error absoluto relativo):** Diferencia absoluta relativa entre los valores esperados y los valores reales, en donde la diferencia de medias se divide por la media aritmética.

. **RSE (Error cuadrado relativo):** Normaliza el error cuadrado de los valores calculados mediante la división por el error cuadrado total de los valores reales.

. **R2 (Coeficiente de determinación):** Nos muestra el valor predictivo del modelo como un valor situado entre 0 y 1, en donde el valor nulo nos indica que nos enfrentamos a un modelo totalmente aleatorio y la unidad muestra un ajuste perfecto. Es importante identificar valores extremadamente altos que pudieran corresponder a una situación de sobre ajuste no deseable.

	MAE	RSME	RAE	RSE	R2
Regresión Lineal	433,678236	551,921396	0,974446	0,939573	0.060426
Decision Forest Regression	174,672966	306,975696	0,392479	0,290659	0,709341
Boosted Decision Tree Regresión	185,178721	291,222958	0,416084	0,261594	0,738406

Figura 19: Tabla de análisis del modelo

Tras transformar los datos mediante la selección de filas y columnas adecuadas, proceder a su split en distintos dataset que nos permitan utiliza el ochenta por ciento para entrenamiento y un veinte por ciento de testeo, entrenamos los modelos para, tras su evaluación, tomar la decisión de que el Boosted Decicison Tree Regresion nos permite unos resultados mas ventajosos, ofreciendo el R2 mas elevado de los tres modelos, un error cuadrático medio mínimo respecto al resto de modelos, un error absoluto relativo mínimamente mas elevado que el árbol de decisión y un error cuadrado menor.

Este será el modelo que utilizaremos para crear un web service que automatizará el proceso no solo de obtención de los parámetros de salida, sino también de tratamiento previo de la información de forma automatizada.

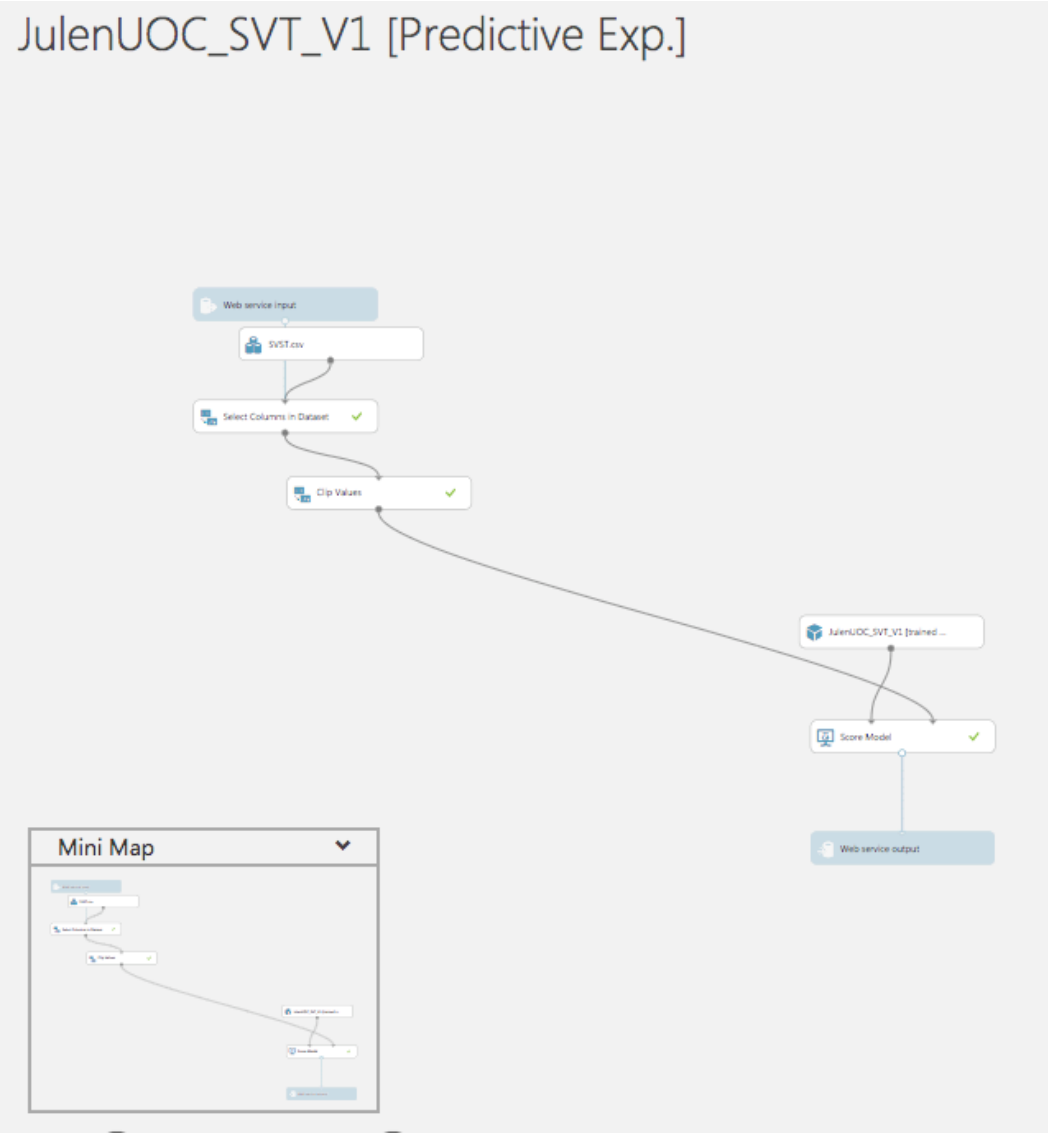


Figura 20: Servicio Web Sensor virtual

3.2. Detección de anomalías energéticas

3.2.1. Objetivo del modelo

Mediante la creación de modelo de anomalías energéticas podemos alertar de un posible comportamiento anómalo que nos permita detectar de forma anticipada consumos energéticos no adecuados en base a parámetros de detección de sensores en equipos o maquinaria.

Detectado el comportamiento anómalo en tiempo real puede corregirse la configuración o trabajo de los equipos hasta recuperar unas condiciones de trabajo con un consumo adecuado.

En este caso particular estamos trabajando con el horno de secado de las tuberías obtenidas tras su paso por las piscinas de tratamiento donde hemos utilizado los sensores virtuales.

Este tipo de hornos, de elevado consumo pueden sufrir comportamientos anómalos que provoquen un coste energético excesivo, por lo que el ahorro que se genera mediante la utilización de tecnologías de inteligencia artificial es rápidamente detectable.

3.2.2. Datos de entrada

Disponemos de siete sensores que recogen información del horno de la linea de producción y una última columna que muestra si la configuración junto al consumo energético dado corresponde a un valor anómalo o a un valor natural y lógico con el proceso observado.

No disponemos de información específica de cada uno de los sensores, por lo que deberemos actuar “a ciegas” en lo que a lógica de negocio se refiere.

- *Sensor A:* Valor numérico entero entre 4 y 72 con un valor medio de 20,903 y una mediana de 18. Su desviación standard es de 12,0588 y dispone de 33 valores únicos.

- *Sensor B:* Valor entero comprendido entre el 250 y el 18424, con una media de 3271,258 y una mediana de 2319,5. Su desviación estándar es de 2822,73.

- *Sensor C:* Detecta cuatro único valores, comprendidos entre el uno y el cuatro, mas propio de una detección de estados o configuración de un equipo determinado. Esta situación se repite con el Sensor D y F.

- *Sensor E:* Valor entero comprendido entre el 19 y el 75, con una media de 35,546 y una mediana de 33.

- S*ensor G:* Dos únicos valores, comprendidos la mayoría de ellos en el primer valor, propio de nuevo de una configuración de equipo o estado.

- ***Label:*** Valor que nos informa según posea el uno o el dos de su comportamiento natural o anómalo.

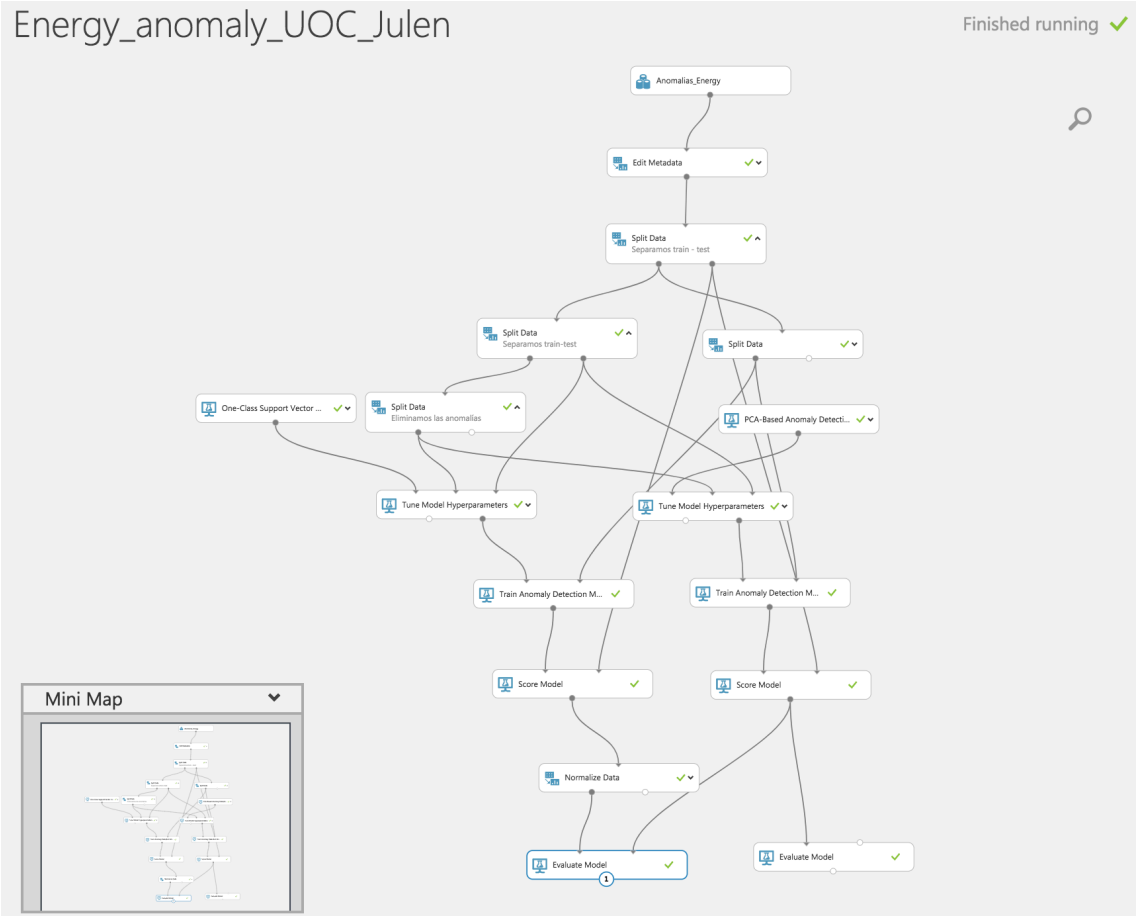


Figura 21: Panel ML Studio detector de anomalías energéticas

3.2.3. Modelos utilizados

Utilizaremos dos modelos que permiten con éxito detectar anomalías en entornos similares a los ofrecidos por nuestro Dataset. Debemos recordar que no contamos con valores de tiempo, por lo que no sería correcto aplicar modelos de series temporales que nos permitan detectar tendencias anómalas en el comportamiento, y presuponer que cada fila mantiene un orden temporal respecto al precedente y posterior sería considerar un modelo basada en una hipótesis no contrastada que puede suponer el fracaso del mismo.

Por lo tanto, modelos como el SVM y PCA, que no están basados en series temporales parecen óptimos para la

resolución de este problema, donde disponemos de datos con distribuciones normales.

Nos encontramos por tanto ante un problema de aprendizaje supervisado, donde conocemos cual debe ser nuestro dato objetivo o de salida y lo debemos englobar dentro de los métodos de clasificación, que en este caso concreto corresponderá a un tipo de clasificación binaria.

El entrenamiento y clasificación en SVM es muy eficiente y muestra un bien funcionamiento en problemas típicos donde no se requiere de un gran coste computacional. Es un modelo robusto y fiable, óptimo para situaciones no particulares o especiales.

El PCA lo hemos podido utilizar bajo suposición de linealidad ante una distribución de los datos como la actual, de tipo Gaussiano.

- **SVM (Support Vector Machine):** Este modelo, aplicado principalmente en escenarios donde la mayoría de datos pueden considerarse normales o estándar es especialmente útil.

Este tipo de modelos son útiles tanto en entornos de clasificación como de regresión. Básicamente el modelo mapea en el espacio los resultados englobados tanto de una propuesta como de otra para, posteriormente, en función del espacio en el que caiga la nueva observación poder predecir su valor o clasificación en cada uno de los grupos asociados.

Como característica especial de este tipo de modelos debemos indicar que se entrenan exclusivamente con los valores que denotan un comportamiento normal, identificando cualquier valor que caiga fuera de ese escenario como valor anormal.

De nuevo, como en ejemplos anteriores, hemos determinado sus parámetros en base a un muestreo de parámetros que nos permita obtener la configuración óptima de los mismos. Este rango de parámetros determina los mejores valores de N y Epsilon, siendo N el valor que representa el límite superior de valores atípicos y Epsilon el valor que determina la tolerancia a la detección, que controla directamente el número de iteraciones utilizadas al optimizar el modelo.

- **PCA (Análisis de componentes principales):** Este modelo se basa en el análisis de la estructura interna de los datos y las posibles relaciones existentes entre los mismos. Se buscan correlaciones entre las variables y determina la combinación de aquella información que identifica variaciones de la mejor forma existente. Utilizando estos valores mas relevantes se crean una serie de funciones que son conocidas como componentes principales.

Dentro de la configuración de los parámetros podemos identificar el número de componentes que utilizará PCA y el número de sobremuestreo que utilizaremos para compensar el desequilibrio existente entre los datos que afectan a un comportamiento normal respecto a los que afectan a un comportamiento anómalo.

3.2.4. El Modelo de Hiperparámetros.

Como parte del entrenamiento del modelo, y puesto que hemos seleccionado un rango de parámetros para su configuración, disponemos de una herramienta que busca la configuración óptima para el entrenamiento del modelo.

Es habitual que requiramos de diversos ajustes en el mismo instante temporal, lo que nos obliga a correr el algoritmo para cada una de las configuraciones posibles. Otra forma sería la de encontrar una metodología de ajuste automático que permita minimizar el error sin requerir de un coste computacional excesivo. SVM nos permite desarrollar este tipo de modelo de hiperparámetros. Existen distintos sistemas como pueden ser la búsqueda en malla ó búsqueda en linea entre otros. Azure nos abstrae de la selección y configuración de este modelo, permitiendo su configuración y gestión óptima automatizada.

Mediante esta herramienta logramos incorporar en un rango determinado de configuraciones aquellas que mejor cumplen el modelo. Para ello debemos introducir los valores sin anomalías junto a la parte de test que utilizaremos para compensar los modelos, que en situaciones de clasificación pueden alterar los resultados debido a los pesos aplicados en el estudio del modelo.

Una vez seleccionados los parámetros de referencia óptimos para el modelo, lo entrenaremos con la totalidad de los datos y evaluaremos.

3.2.5. Evaluación de los modelos

Los métodos de evaluación en procesos de clasificación se determinan por los siguientes valores:

- *Accuracy:* Mide la bondad del modelo como la proporción de resultados verdaderos respecto al total de casos existentes
- *Precision:* Proporción de resultados positivos respecto al número total de resultados positivos existentes
- *Recall:* La fracción de todos los resultados correctos que son devueltos por el modelo
- *F-Score:* Promedio ponderado de precisión y recuperación entre el valor nulo y la unidad, siendo el valor óptimo de 1.
- *AUC:* Area de trabajo bajo la curva trazada con verdaderos positivos en el eje y, y de falsos positivos en el eje x.
- *Average log loss:* Pérdida promedio de registro se utiliza para expresar la penalización de resultados incorrectos, calculada como la diferencia de dos distribuciones de probabilidad, la verdadera y la del modelo.
- *Training log loss:* Nos muestra la incertidumbre del modelo al comparar las probabilidades que genera con los valores conocidos.

Figura 22 Servicio web detección de anomalías energéticas

	SVM	PCA
True positive	10	16
False Negative	53	47
False Positive	17	39
True Negative	120	98
Accuracy	0,650	0,570
Recall	0,159	0,254
F1 Score	0,222	0,271
Threshold	0,5	0,5
AUC	0,496	0,496
Precision	0,370	0,291

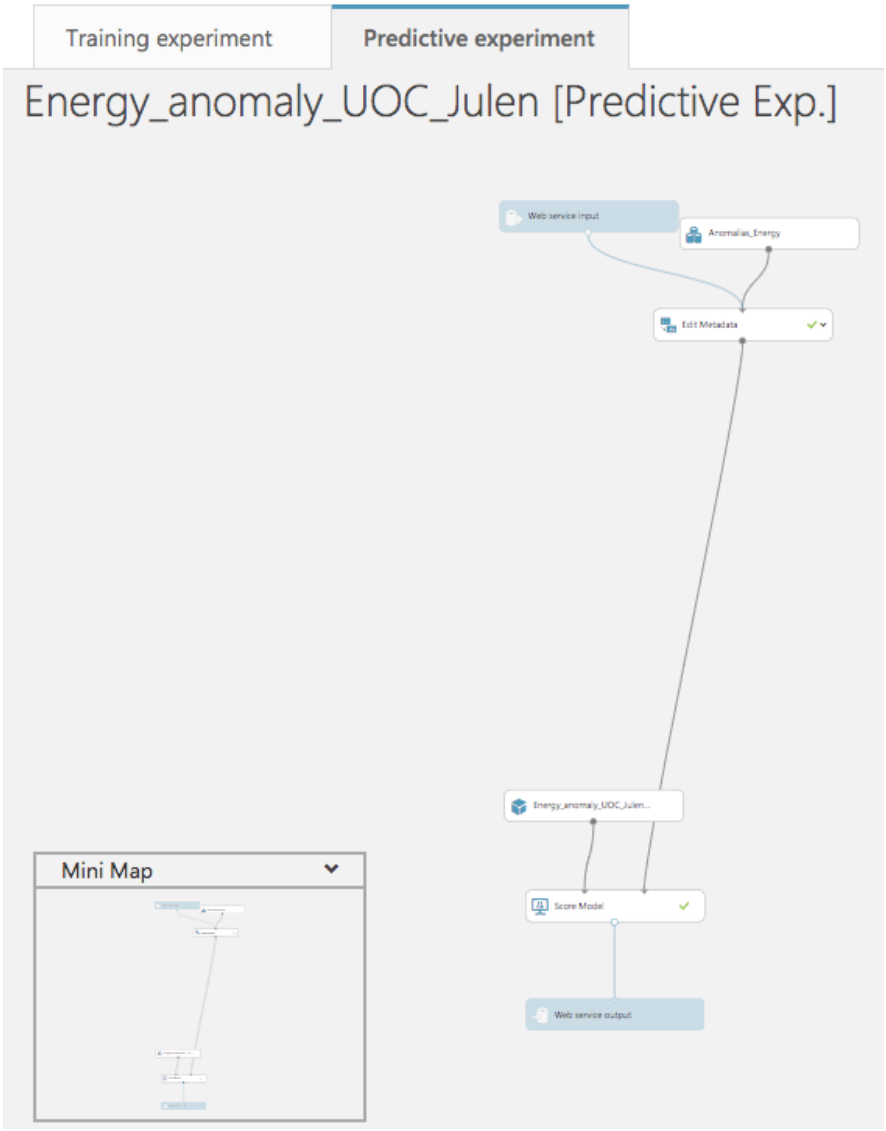


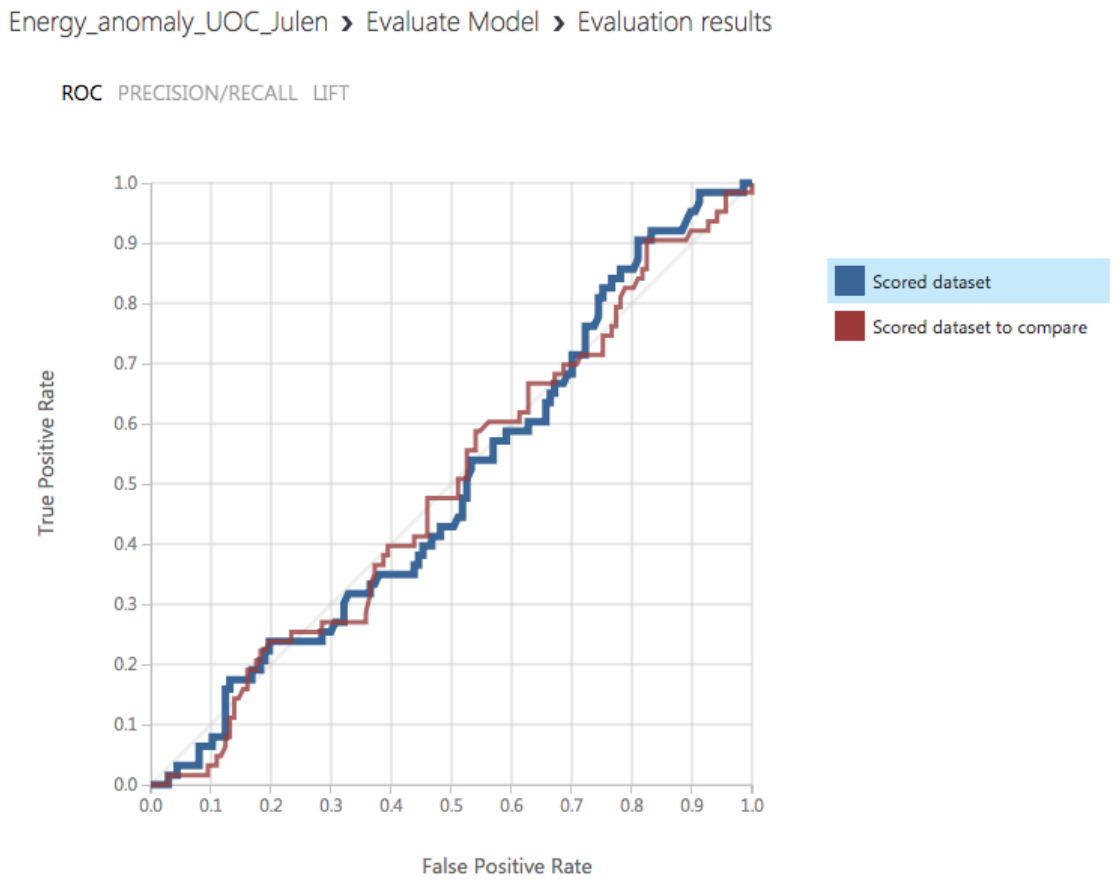
Figura 23: Tabla de análisis del modelo energético

Figura 24: Evaluación comparada de la matriz de confusión

El SVM, ante los mismos parámetros de entrada alcanza mejores resultados de validación, por lo que podemos seleccionar dicho modelo para crear nuestro web service sobre el problema presentado

3.3. Modelo de calidad

Nos encontramos ante la necesidad de crear un nuevo modelo de clasificación, en este caso con el objetivo de determinar si un tubo laminado debe ser enviado a revisión



por prever problemas de calidad con el mismo a la finalización del proceso productivo.

La resolución de problemas de detección de fallos de calidad mediante el uso de analítica avanzada de datos es uno de los extremos mas utilizados en el sector de la Industry 4.0 puesto que permite crear una barrera de selección sobre el producto final basado en inteligencia artificial, y no sobre un muestreo aleatorio.

En este campo, se han desarrollado sistemas de visión artificial mediante la utilización de redes neuronales convolucionales, que en este caso concreto, si bien se han utilizado redes neuronales para su establecimiento, no se han desarrollado para dar servicio a un Dataset de visión artificial, sino a un sistema de control de calidad basado en parámetros de medición y configuración de los equipos previos del proceso productivo.

3.3.1. Modelos utilizados

- **Two-Class Neural Network:** Este modelo sirve para predecir un valor objetivo que dispone exclusivamente de dos valores de salida.

Una red neuronal es un conjunto de capas que se encuentran interconectadas entre si que se excitan en función de unos parámetros y pesos de entrada, obteniendo una salida, en este caso binaria, según las excitación de las capas precedentes.

Los valores configurables de la red neuronal son los siguientes:

- *Número de nodos ocultos* que conforma la capa de la red
- *Tasa de aprendizaje*, tomando como tal el valor del paso en cada acción, modificando con ello el tiempo de convergencia
- *Número de iteraciones* de aprendizaje
- *Diámetro de pesos de aprendizaje inicial*, o punto de partida de peso de los nodos en el sistema
- *Impulso o peso* que se aplica durante el aprendizaje a los nodos de iteraciones previas
- *Método de normalización* de los valores para contener el peso en la red.

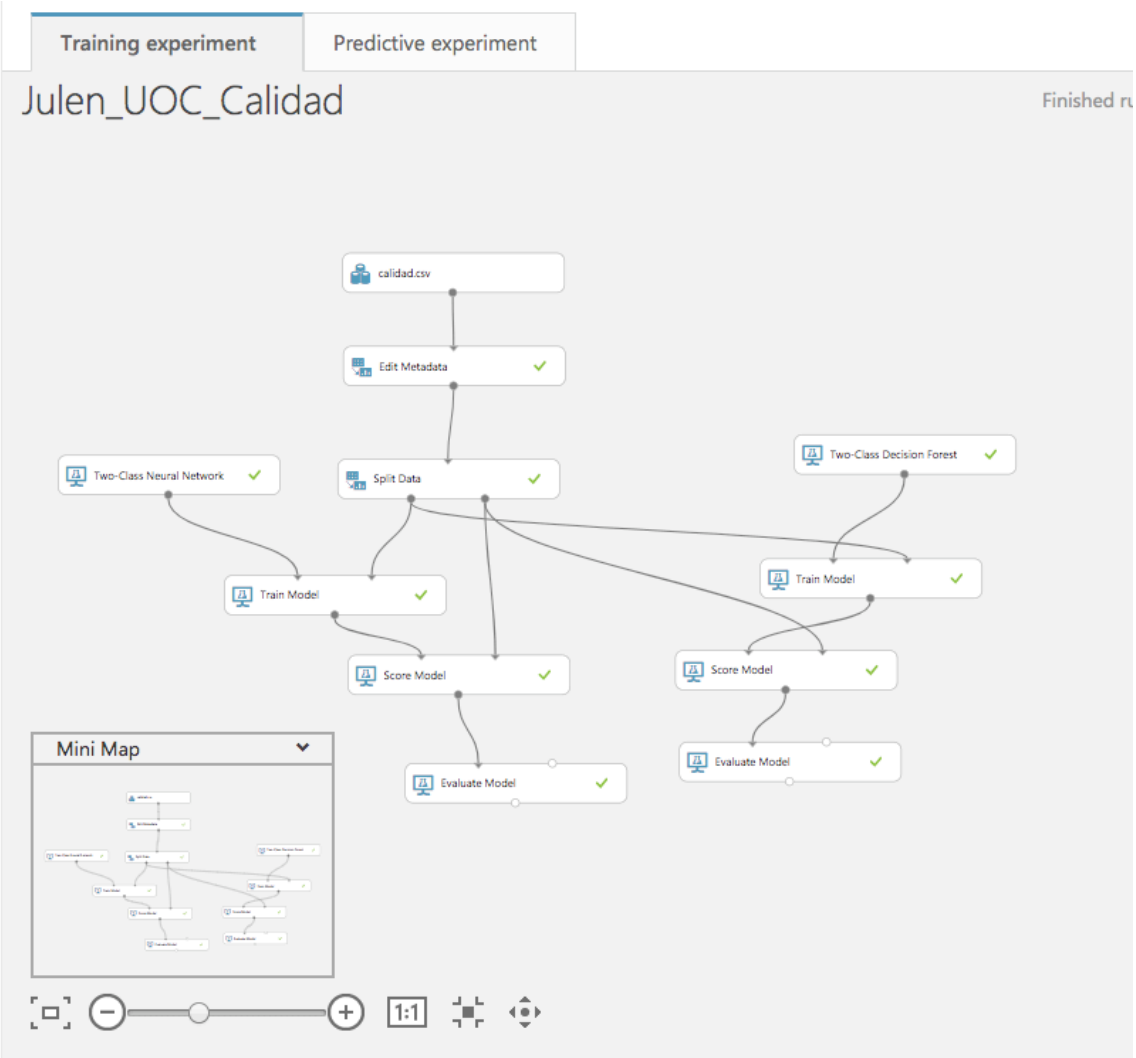


Figura 25: Panel ML Studio de calidad

Two Class decision Forest: Es un algoritmo de bosque de decisión integrado en los métodos de ensemble, donde se crean múltiples árboles de decisión y se toma una salida en base a la salida mas repetida del 'bosque'.

3.2.5. Evaluación de los modelos

La evaluación de los modelos se basa en un sistema de información similar al apartado anterior, en donde realizábamos clasificación para obtener puntos anómalos

	RED NEURONAL	DECISION FOREST
True positive	41	36
False Negative	109	114
False Positive	8	11
True Negative	842	839
Accuracy	0,883	0,875
Recall	0,273	0,240
F1 Score	0,412	0,365
Threshold	0,5	0,5
AUC	0,749	0,696
Precision	0,837	0,766

Figura 26: Tabla de análisis del modelo de calidad

La red neuronal ofrece mejores resultados de precisión y bondad del modelo, si bien quizá este sea un punto importante para comentar los datos ofrecidos por la matriz de confusión.

En esta matriz podemos observar visualmente las observaciones de cada clase y su respuesta tanto acertada como fallada según el resultado de salida real o esperado.

La red neuronal tiene mayor número de falsos positivos, es decir, nos informa de que existe un fallo de calidad en mas

situaciones de las reales que el decisión forest, pero tiene un menor número de falsos negativos.

En un problema de calidad es importante destacar que es preferible equivocarse con un falso positivo y tener que revisar un tubo que se encontraba en perfectas condiciones de uso que no dar salida a un tubo que no se encuentra en condiciones de salida al mercado con un falso negativo.

Por ello, a parte de los datos que muestren la precisión y bondad del modelo debemos siempre considerar cual es la salida mas acertado del error, es decir, si nuestro modelo falla, donde preferimos que falle.

La red neuronal por lo tanto cumple todos los requerimientos que nuestro sistema de detección de calidad necesita.

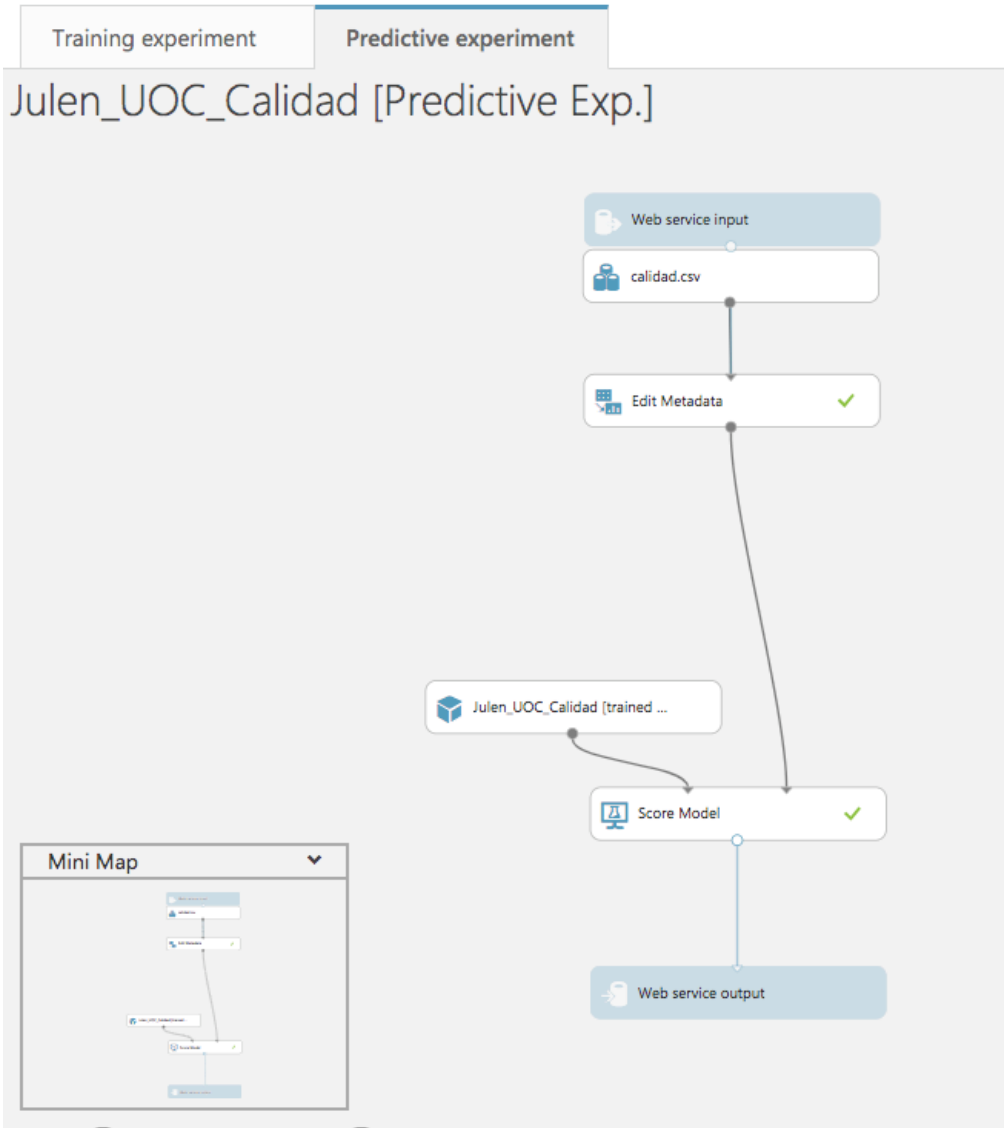


Figura 27: Servicio web calidad

4. Modificación y adaptaciones de la arquitectura a nuestra inserción de información

4.1. Modificación del código Java de emulación

Tomando como punto de partida el código funcional en Java descrito en el apartado 2.3 del presente proyecto, debemos proceder a su modificación para que nuestro paquete de envío incluya las variables de entrada de nuestros modelos analíticos.

Cada uno de los modelos requiere de unos parámetros determinados que hemos incluido en un array de valores independiente: '**sv**' para los valores que permitirán calcular el dato de sensor virtual, '**en**' para los valores que permiten detectar una anomalía energética y '**ca**' para aquellos que permiten detectar problemas de calidad en las piezas.

Estos valores que se incorporan a nuestro objeto de telemetría, con el fin de que no envíen datos de forma secuencial se incorpora un orden de envío según una variable entera que recorre las 20 posibles filas de datos en cada uno de los arrays con carácter aleatorio.

4.2. Creación de funciones y nuevos outputs en Job Streaming

La creación de las querys es uno de los puntos mas importantes para el correcto funcionamiento del sistema. Previo a su desarrollo creamos las salidas necesarias de la información obtenida a través tanto de los modelos creados como de la información recogida de los sensores emulados.

Los puntos de salida son los siguientes (Outputs):

- *bloboutputca*: Salida de los datos relativos al entorno de calidad hacia un blobstorage que alberga el contenedor contanierca.

- *bloboutputen*: Salida de los datos relativos al entorno de energía hacia un blobstorage que alberga el contenedor bloboutputen.

- *bloboutputsv*: Salida de los datos relativos al entorno del sensor virtual hacia un blobstorage que alberga el contenedor svcontainer.

- *outputBICalidad*: Salida de los datos tanto emitidos por los sensores como calculados hacia el PowerBI en tiempo real relativos al entorno de calidad.
- *outputBIEnergy*: Salida de los datos tanto emitidos por los sensores como calculados hacia el PowerBI en tiempo real relativos al entorno de energía.

- *outputBISV*: Salida de los datos tanto emitidos por los sensores como calculados hacia el PowerBI en tiempo real relativos al entorno del sensor virtual.

- *outputcontador*: Salida de información relativa al número de piezas fabricadas y cantidad de bloques de información enviada. Se remite directamente a PowerBI.

Las querys en un Job Streaming se encargan de enlazar los datos de la entrada o input, con cada una de las salidas, pudiendo en el camino transformar la información en función de nuestros intereses.

En nuestro caso específico la transformación que realizaremos con los datos será el paso por cada uno de los webservice creados en los modelos de entrenamiento, de forma que el input del modelo será cada bloque de información recogido de los sensores en tiempo real y la salida será el output calculado de cada uno de los modelos.

Estos modelos de machine y deep learning creados son considerados en el entorno de Azure como **funciones**, a las cuales les otorgamos unos parámetros de entrada y nos devuelven un parámetro de salida.

Hemos creado, dentro del mismo bloque de Stream Analytics una función para cada uno de los modelos analíticos:

- *uoc_calidad*: requiere 6 parámetros de entrada y trabaja sobre el modelo de calidad

- *uoc_energy*: requiere de 8 parámetros de entrada y trabaja sobre el modelo energético

- *uoc_svt*: requiere de 9 parámetros de entrada y trabaja sobre el modelo de sensor virtual

Las querys por lo tanto quedarán descritas de la siguiente forma:

```
WITH data AS (SELECT * FROM inputIot),
```

Uno de los handicaps que tiene el sistema de Streaming de Azure es la incapacidad para trabajar con múltiples fuentes de entrada dentro del sistema de querys y selección de información, por lo que requiere de la unificación de los paquetes de información en un único conjunto por cada timestamp, que en este caso hemos denominado 'data'. A partir de 'data', que engloba toda la información, se comienza a trabajar con ella.

En este punto determinamos una nueva función que incorpora incluso los datos calculados en cada función asociada al modelo

```
funcionSV AS(
            SELECT   CE,   Temp_CE,   PH,   Temp_PH,   Turbidez,   Voltaje,
EventEnqueuedUtcTime,EventProcessedUtc,
     uoc_svt ( deviceId   , CE, Temp_CE , PH , Temp_PH, Turbidez, Voltaje,
DisPlanta  , eventTime)
    AS resultsv FROM data
    ),

funcionCALIDAD AS(
     SELECT contador_piezas_total, CA_cambio , CA_MOSE , CA_MOS5, CA_MES1,
CA_MES2, CA_MES3, EventEnqueuedUtcTime,EventProcessedUtc,
    uoc_calidad ( CA_cambio , CA_MOSE , CA_MOS5, CA_MES1, CA_MES2, CA_MES3 )
    AS resultca FROM data
    ),

funcionENERGY AS(
```

```
      SELECT Col2, Col5 , Col8, Col11, Col13, Col16, Col18, Col21 ,
EventEnqueuedUtcTime,EventProcessedUtc,
    uoc_energy ( Col2, Col5 , Col8, Col11, Col13, Col16, Col18, Col21 )
    AS resulten FROM data
     )
```

Seleccionamos de cada función creada la información que nos parece interesante y la remitimos a cada salida, tanto de PowerBI como del BlobStorage

```
SELECT     CE,     Temp_CE,     PH,     Temp_PH,     Turbidez,     Voltaje,
EventEnqueuedUtcTime,EventProcessedUtc,
CAST (resultsv.[Scored Labels] AS float) AS resultadosv
INTO outputBISV
FROM
funcionSV

SELECT     CE,     Temp_CE,     PH,     Temp_PH,     Turbidez,     Voltaje,
EventEnqueuedUtcTime,EventProcessedUtc,
CAST (resultsv.[Scored Labels] AS float) AS resultadosv
INTO bloboutputsv
FROM
funcionSV

SELECT   CA_cambio   ,   CA_MOSE   ,   CA_MOS5,   CA_MES1,   CA_MES2,   CA_MES3,
EventEnqueuedUtcTime,EventProcessedUtc,
CAST (resultca.[Scored Labels] AS float) AS resultadoca
INTO outputBICALIDAD
FROM
funcionCALIDAD

SELECT    contador_piezas_total,  CA_cambio   ,   CA_MOSE   ,   CA_MOS5,   CA_MES1,
CA_MES2, CA_MES3, EventEnqueuedUtcTime,EventProcessedUtc,
CAST (resultca.[Scored Labels] AS float) AS resultadoca
INTO bloboutputca
FROM
funcionCALIDAD

SELECT   Col2,   Col5   ,   Col8,   Col11,   Col13,   Col16,   Col18,   Col21   ,
EventEnqueuedUtcTime,EventProcessedUtc,
CAST (resulten.[Scored Labels] AS float) AS resultadoen
INTO outputBIENERGY
FROM
funcionENERGY

SELECT  Col2,  Col5  ,  Col8,  Col11,  Col13,  Col16,  Col18,  Col21  ,
EventEnqueuedUtcTime,EventProcessedUtc,
CAST (resulten.[Scored Labels] AS float) AS resultadoen
INTO bloboutputen
FROM
funcionENERGY


SELECT contador_piezas_total
INTO outputcontador
FROM data
```

4.2. Comprobación y testeo del sistema

Azure permite monitorizar sus servicios. En este caso el servicio mas importante que nos indica si los datos se encuentran fluyendo en tiempo real con fluidez es el sistema de observación de Job Streaming.

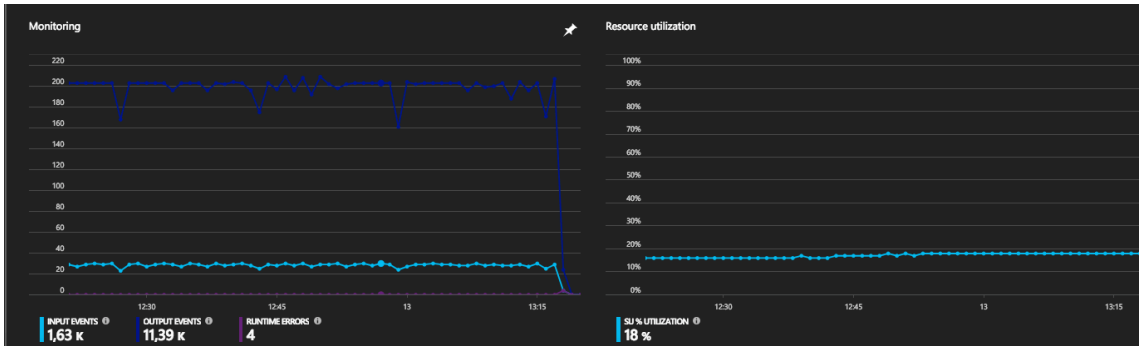


Figura 28: Monitorización del servicio

El azul claro muestra los eventos de entrada, mientras que la línea de color azul oscuro monitoriza los valores de salida. Es importante recordar que en nuestras querys hemos agrupado la totalidad de datos con las que trabajaremos en el valor 'data', por lo que se multiplican los datos de salida respecto a los datos de entrada.

5. Diseño y creación del Business Intelligence a nivel histórico y su integración con el tratamiento de datos según los modelos calculados en tiempo real.

PowerBI permite crear escenarios para visualización de datos tanto en tiempo real mediante sus sistema de BI en

cloud como el análisis básico y visualización por lotes mediante PowerBI Desktop.

Posteriormente pueden enlazarse ambas aplicaciones para configurar una única aplicación funcional que puede ser compartida entre los usuarios con acceso.

En primer lugar detallemos PowerBI en tiempo real. La configuración de la información es fácilmente modificable y adaptable a las necesidades finales de usuario. Esta propuesta es una de tantas que podría haberse realizado, y está enfocada a poder visualizar distintos artefactos en el sistema que pudieran servir como referencia.

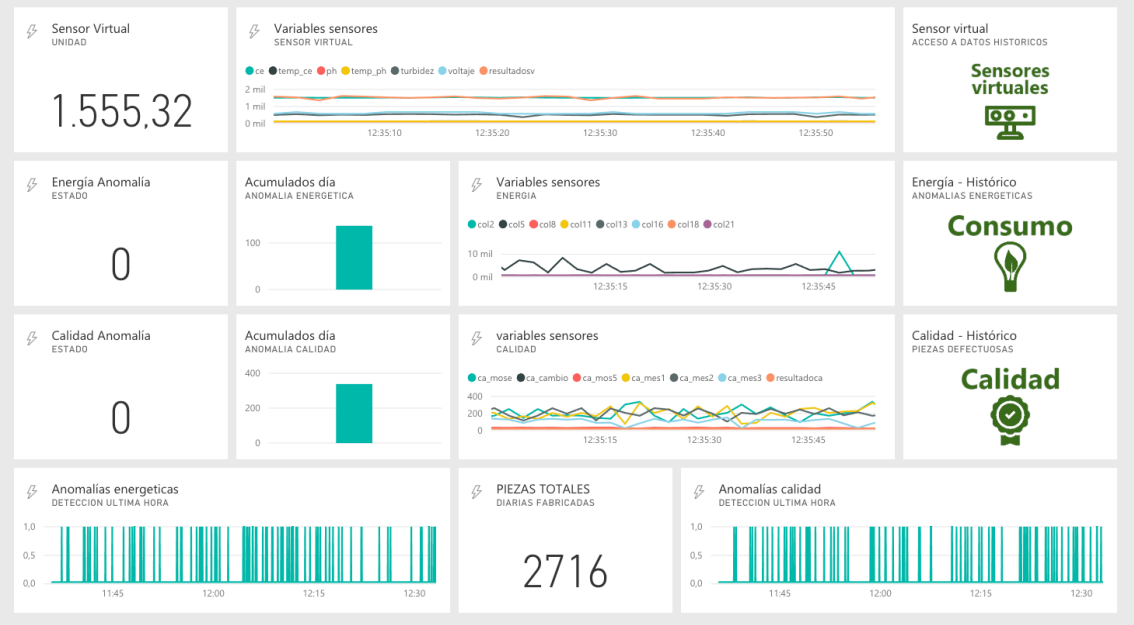


Figura 29: Panel principal de tiempo real

Por un lado tenemos las tarjetas, que muestran un valor numérico. En nuestra aplicación observamos el valor del sensor virtual, la existencia de una anomalía energética (0 no existe, uno si existe), así como la existencia de una pieza con posibles fallos de calidad que debería de enviarse a revisión (0 pieza correcta, uno si la pieza debe revisarse)

Las gráficas temporales incorporan los valores temporales de cada variable estudio en cada instante temporal, y tanto en la parte energética como de calidad se muestra el número de alertas acumuladas por día.

En la última fila observamos con un pico de color verde la existencia de un fallo de calidad o anomalía energética

durante la última hora de producción junto al número de piezas totales fabricadas.

Desde la última columna podemos acceder a los archivos históricos de cada área de estudio.

Esta información no se adquiere desde los output de salida que enfocan hacia PowerBI, sino que se adquieren desde el blobstorage, es decir, el almacén de datos donde hemos ido volcando la información obtenida tanto de los sensores de entrada como de los modelos de salida.

En un entorno real esta información almacenada nos serviría para mejorar nuestros modelos de forma automatizada, si bien, en el caso que nos ocupa, al inyectar información limitada actuar de esta forma no haría mas que limitar los resultados y empeorar el modelo, puesto que solo inyectamos 20 filas de forma repetitiva y alterna.

Al acceder al cuadro de sensores virtuales nos adentramos en el modelo de visualización de business intelligence, donde procuramos convertir los datos en conocimiento.

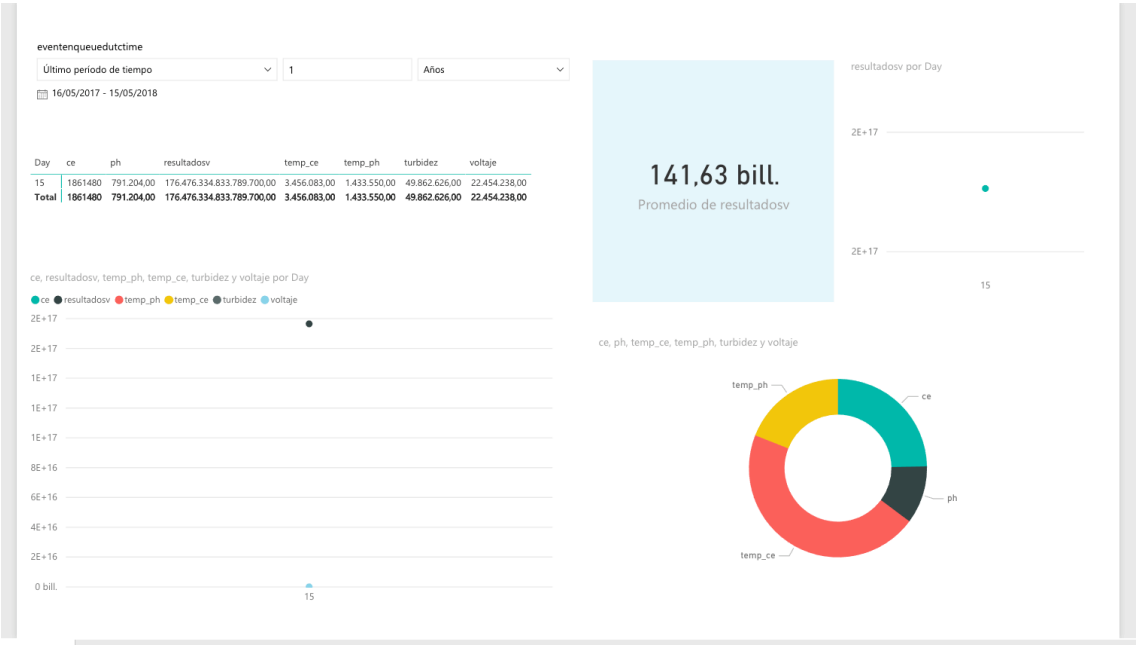


Figura 30: Panel histórico BI sensor virtual

PowerBI Desktop, la aplicación con la que creamos los paneles de información sin tiempo real permite realizar un

pequeño proceso de ETL (Extracción transformación y carga) hasta 10 GB sin necesidad de utilizar otro sistema de ETL externo como pudiera ser un Pentahoo si nos decantamos por un OpenSource o un DataFactory con una base de datos SQL si continuamos dentro del entorno Serverless de Azure.

Recoge la información de cada contenedor de Blob Storage, la transforma según el proceso diseñado y la carga en el BI, todo ello con la capacidad de actualizarse cada hora de forma programada.

En la siguiente ilustración observamos el análisis realizado para el sector de estudio energético

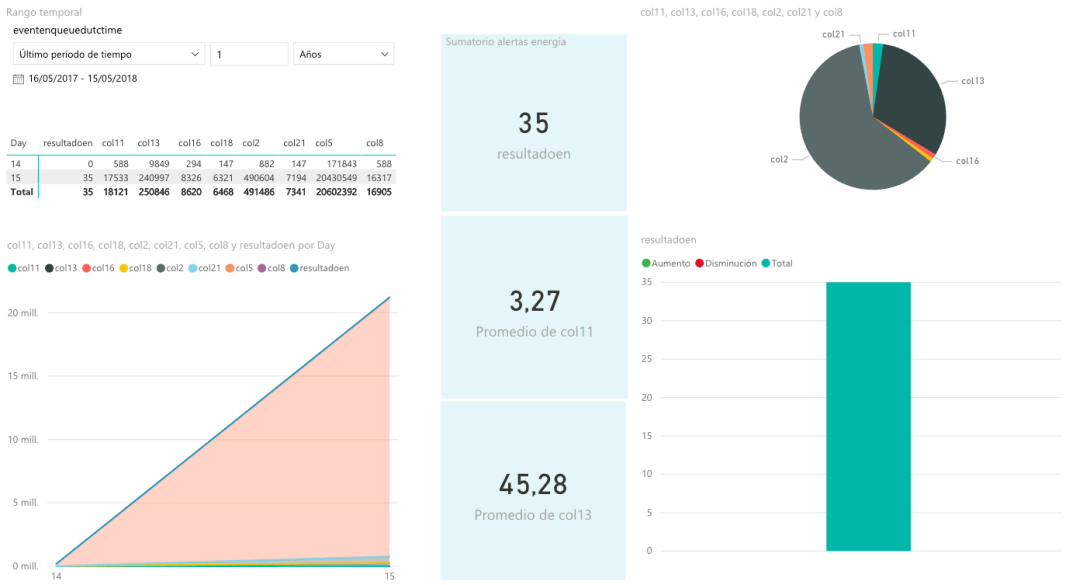
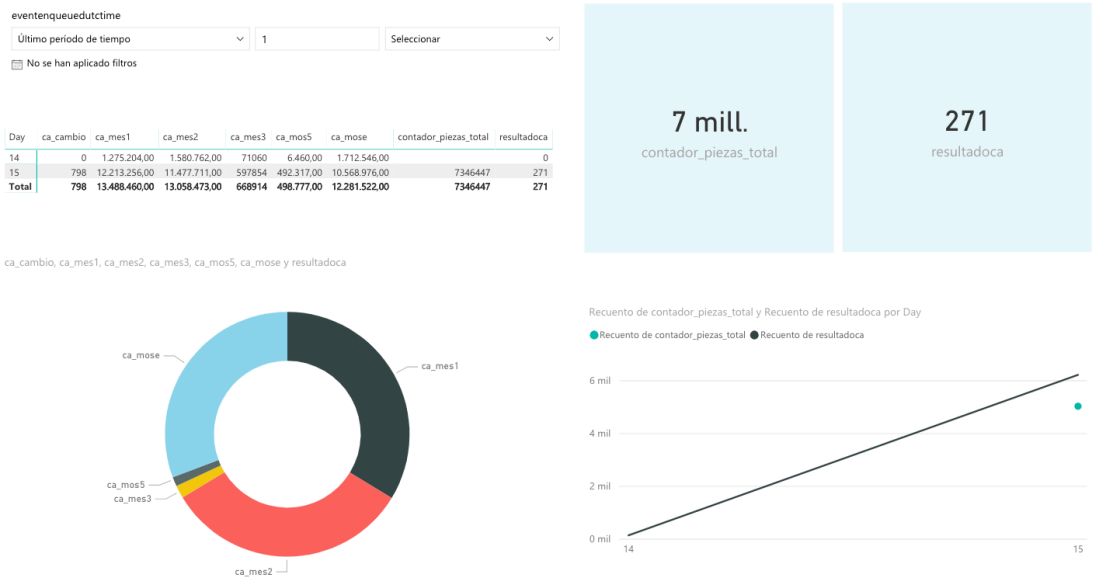


Figura 31: Panel histórico BI Estudio energético

Y en esta última el panel Bi para el área de calidad.

Figura 32: Panel histórico BI calidad

Cada panel se actualiza en función del espectro temporal seleccionado, y cada uno de los gráficos o paneles es



exportable a los formatos mas habituales de trabajo como excel o csv.

6. Escalabilidad del sistema

La misma arquitectura y programación realizada para trabajar con una entrada de 23 datos con una frecuencia de 2 sg, lo que nos aporta 993.600 datos diarios es escalable prácticamente de forma indefinida.

Un único IoTHub es capaz de trabajar con 400.000 mensajes diarios y podemos escalarlo hasta los 300 millones de mensajes para un único IotHub. A partir de esa cifra simplemente necesitamos crear un IotHub nuevo o tantos como nos hagan falta para dar servicio al sistema.

Para calcular la salida de los modelos en tiempo real debemos configurar las máquinas que darán soporte a la computación. Un equipo básico de 4GB de RAM permiten en nuestro caso responder a las consultas realizadas en 296,33 ms con un coste de 2 dólares por hora.

Estos equipos pueden escalar tanto horizontal como verticalmente dando servicio a sistemas que requieran del

tratamiento de una cantidad ingente de información sin necesitar modificar la arquitectura del servicio.

Los job streaming permiten incrementar las unidades de computación asociadas al servicio hasta 120 unidades de computación. En nuestro caso concreto, con casi un millón de datos procesados diariamente, estamos utilizando exclusivamente un 18% de los recursos ofrecidos por una única unidad, por lo que aproximadamente podríamos dar servicio con el mismo Job streaming a 600 millones de datos diarios.

Por último, el alojamiento de la información recibida es también autoescalable, prácticamente sin límites, tanto en BlobStorage como en DataWarehouse habilitados al efecto. Es importante recordar que ante un volumen elevado de información sería preciso crear un proceso adaptado de ETL que diera soporte a los paneles de visualización históricos, para fomentar un rápido acceso a la información.

El coste necesario para un grado de computación y análisis como el del presente proyecto es el siguiente:

. IotHub limitado a 400.000 mensajes por día, con un coste de 21,08 €/mes

. Stream Analytics, una unidad funcional durante todo un mes por 75,29 €/mes

. Machine Learning Studio. Para el área de trabajo y estudio de modelos hasta 10 GB de procesamiento no tiene coste asociado.
Por utilizar la API de conexión para calcular datos de salida según los modelos utilizados tenemos un coste de 84,44 €, incluyendo 100.000 transacciones y 25 horas de proceso.

. Blob Storage por 0,0017 €/GB/mes

. PowerBI, licencia comercial por 8 €/mes, solo requerida para la visualización de información en tiempo real. Para realizar paneles de información por lotes se utiliza PowerBI Desktop sin coste asociado.

Por lo tanto, el presente proyecto tiene un coste asumido de 188,81 €/mes si el producto estuviera funcionando de forma constante en producción.

Puesto que lo dejamos parado y solo se utiliza para el desarrollo y pruebas del trabajo fin de grado el coste no alcanza en ningún caso los 30 €/mes de uso.

7. Conclusiones y ampliaciones del sistema

Diseñar, programar y desplegar un sistema de captación, procesado de información basado en modelos de machine learning, almacenamiento y visualización en tiempo real e históricos mediante un sistema de información tradicional hubiera requerido de un esfuerzo titánico prácticamente inalcanzable para un único individuo en un plazo similar al existente para desarrollar el presente trabajo fin de grado.

El esfuerzo requeriría de la especialización tanto en sistemas, comunicaciones y sockets, sistemas operativos ó machine y deep learning, a parte de un fuerte conocimiento en programación tanto en Python o R para la creación de los modelos como Js para la parte servidor pensando en un sistema basado en NodeJS, JS para la parte front mediante el uso de D3.js, HTML y CSS y un espacio backend para la visualización por ejemplo en PHP.

Este puede ser un modelo de tantos otros. Cualquiera que elijamos probablemente requiera de un esfuerzo y conocimiento de diversas disciplinas que pocas personas aúnan en un único individuo.

Si un sistema como el descrito requiere de una escalabilidad en producción importante deberemos desechar la mayoría del trabajo realizado y comenzar a utilizar openSource ligado a los entornos de Big Data como podrían ser servicios de Kafka, Spark Streaming, Hbase como espacio de alojamiento todo ello bajo el soporte onpremise de sistemas como Cloudera o HortonWorks que requieren de una inversión en equipos de computación para su despliegue en arquitecturas horizontales que incrementan los costes de

desarrollo y despliegue hasta puntos que convierten el proyecto en algo insostenible.

Esta ha sido la razón de la caída de muchos proyectos de analítica de datos que no han podido llevarse a cabo, tanto por costes de inversión y mantenimiento como por costes humanos y de captación de talento.

En este trabajo fin de grado he podido desarrollar, una única persona, una arquitectura completa de captación y tratamiento de datos en tiempo real mediante la utilización de servicios Serverless, en este caso de Azure, pero que es extrapolable a cualquier otro proveedor Cloud como Google o AWS.

Los tiempos empleados en el desarrollo del proyecto han sido adecuados y adaptados a las necesidades reales del mismo, alcanzando la totalidad de los objetivos marcados, y pudiendo disponer de un servicio en ejecución completo y que muestra un flujo de trabajo real propio de un entorno de Industry 4.0 donde se desean adoptar decisiones en base a sistemas de aprendizaje computacional basado en la analítica de datos.

Es obvio que el presente proyecto se basa en datos que no pertenecen a un entorno real, y por lo tanto se carece del conocimiento de negocio que debería de aportarse en un proyecto real, y que dota de inteligencia a los datos, convirtiendo esa información en decisiones. Por lo tanto, la parte de BI histórica visualizada muestra una posibilidad de expresión mas que una necesidad real basada en un caso o modelo de negocio, que es como debería de sustentarse un proceso de visualización que de soporte a las decisiones funcionales de la empresa.

Un aspecto a destacar es la capacidad de escalabilidad del proyecto, prácticamente infinita a costa de incrementar los costes de mantenimiento y computación, siendo este proceso totalmente transparente para el usuario.

Dentro de los objetivos del trabajo final de grado se encontraba la creación de un emulador de sensores capaz de, utilizando las librerías adecuadas de JAVA emular el comportamiento de distintos sensores, diseñar e implementar la arquitectura completa de computación capaz tanto de procesar los datos en tiempo real como de computarlos, crear los modelos analíticos necesarios para resolver los distintos problemas de sensorización virtual, calidad y eficiencia energética, desarrollar e implementar un BI o cuadro de mandos que permita dotar la decisión de

información y almacenar toda la información entrante en una base de datos no estructurada para su posterior análisis y mejora de los procesos.

Cada uno de los objetivos marcados se ha resuelto a lo largo del presente trabajo, realizando la implementación y simulación del mismo modo que se hubiera realizado en un entorno industrial real y confirmando el potencial de los servicios Serverless del Cloud para el desarrollo de proyectos futuros de inteligencia artificial.

Será parte de un futuro trabajo de inteligencia artificial dotar de reacción a las respuestas de los modelos, cerrando el ciclo con un sistema que permita prescindir de la decisión humana en aquellas acciones que el modelo recomiendo adoptar para mejorar la eficiencia del proceso.

En el presente trabajo la información recogida por los sensores es visualizada en un cuadro de mandos en tiempo real, (similar a un sistema SCADA industrial), a la vez que mostramos los datos calculados por nuestros modelos analíticos, capaces de realizar un proceso predictivo, pero nuestro sistema simplemente alerta de una circunstancia al responsable de producción. Es decir, es éste último quien debe tomar la decisión sobre como actuar sobre los equipos, y si bien hemos avanzando y mejorado en sobremanera el proceso productivo al dotar de relevante información del proceso en el mismo instante en el que este se produce, además de permitir una monitorización remota de múltiples plantas, aún ‘perdemos’ tiempo en la toma de decisión humana.

La misma arquitectura creada para desarrollar este proyecto puede emplearse para enviar información que actué de acuerdo a los parámetros óptimos de configuración de los equipos, pero en este caso actuando contra los PLCs industriales que tienen la capacidad de modificar y actuar contra los equipos productivos.

Este último paso nos permite disponer de una fábrica autónoma con un elevado grado de eficiencia y capaz de mejorar y aprender a cada paso en el desarrollo de sus sistemas productivos.

8. Glosario

AWS: Plataforma de servicios web propiedad de Amazon para dar servicios en la nube alojados en los DataCenter de la compañía

Azure: Plataforma de servicios web propiedad de Microsoft para dar servicios en la nube alojados en los DataCenter de la compañía

Blobs: Base de datos NoSQL ofrecida como servicio por Azure que se compone de 'contenedores' independientes entre si y permiten alojar datos en formatos no estructurados.

Business Intelligence: También denominado inteligencia empresarial o inteligencia de negocio es el conjunto de estrategias, aplicaciones, datos, productos y tecnologías destinadas a la creación de conocimiento a través del análisis de los datos existentes de la empresa.

Google Platform: Plataforma de servicios web propiedad de Google para dar servicios en la nube alojados en los DataCenter de la compañía

IotHub: Servicio Serverless de Azure que permite actuar como desacoplador de arquitecturas para la recolección de datos provenientes de sensores o equipos externos al sistema.

PowerBI: Sistema de visualización ofrecido por Microsoft que permite bajo suscripción mostrar información y crear procesos de extracción, transformación y carga basada en datos alojados en servicios propios o de terceros.

Streaming Job: Servicio Serverless de Azure que permite recoger y transformar datos desde distintos puntos o servicios tanto de Azure como externos hasta alojarlos en servicios propios o de terceros.

9. Bibliografía

[1] Prabhanjan Narayanachar Tattar (2013). R Statistical Application Development by Example Beginner's Guide

[2] Angel Luis Castellanos Peñuela (2013). Tesis Doctoral: Algoritmos para minería de datos con redes neuronales

[3] Manpreet Singh Arshad Ali (2016). Big Data Analytics with Microsoft HDInsight in 24 hours, Part IV.

[4] Sebastian Raschka, (2015). Python Machine Learning.

[5] Yvone Gala García (2013), Algoritmos SVM para problemas sobre Big Data (Master en investigación e innovación en Tecnologías de la información y las comunicaciones).

[6] Documentación Microsoft Azure web. https://docs.microsoft.com/es-es/azure/

[7] Documentación Azure ML Studio https://docs.microsoft.com/es-es/azure/machine-learning/studio/

[8] Documentación IotHub https://docs.microsoft.com/es-es/azure/iot-fundamentals/ https://docs.microsoft.com/es-es/azure/iot-hub/

[9] Documentación ML Service https://docs.microsoft.com/es-es/azure/machine-learning/service/

[10] Documentación Azure Functions https://docs.microsoft.com/es-es/azure/azure-functions/

[11] Documentación Stream Analytics https://docs.microsoft.com/es-es/azure/stream-analytics/

[12] Documentación PowerBI https://docs.microsoft.com/es-es/power-bi/

10. Anexo I: Código final JAVA emulador

// Importamos los paquetes que nos permitirán utilizar las librerías destinadas a la conectividad con AZURE

```
package com.mycompany.app;
import com.microsoft.azure.sdk.iot.device.*;
import com.mycompany.app.App.TelemetryDataPoint;
import com.google.gson.Gson;

import java.awt.List;
import java.io.*;
import java.net.URISyntaxException;
import java.security.Timestamp;
import java.util.ArrayList;
import java.util.Random;
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;
public class UOC
{


        // Variable estática que contendrá el String de conexión con el dispositivo creado en el IOT Hub
        private static String connString = "HostName=iotUOCJulen.azure-devices.net;DeviceId=uocDevice;SharedAccessKey=ko2nLgWoYBpX9dhhzfGo7dxYrpvMKQwyEHF/hr+0OQ4=";


        //Determinamos el protocolo de comunicación, en este caso seleccionamos MQTT
        private static IotHubClientProtocol protocol = IotHubClientProtocol.AMQPS_WS;
        private static String deviceId = "dispositivoUOC";
        private static DeviceClient client;

        // La clase TelemetryDataPoint nos permitirá determinar atributos que se relacionarán con cada sensor
        // para ser posteriormente empaquetados y enviados junto con sus identificadores de servicio
        private static class TelemetryDataPoint {

                public String deviceId;
                public double CE;
                public double Temp_CE;
                public double PH;
                public double Temp_PH;
                public double Turbidez;
                public double Voltaje;
                public String DisPlanta;

                public double CA_Cambio;
                public double CA_MOSE;
                public double CA_MOS5;
                public double CA_MES1;
                public double CA_MES2;
                public double CA_MES3;

                public double Col2;
                public double Col5;
                public double Col8;
                public double Col11;
                public double Col13;
                public double Col16;
                public double Col18;
```

```
        public double Col21;


        public int contador_piezas_total = 0;



        public String serialize() {
          Gson gson = new Gson();
          return gson.toJson(this);
        }
      }
    private static class EventCallback implements IotHubEventCallback {
        public void execute(IotHubStatusCode status, Object context) {
          System.out.println("IoT Hub responded to message with status: " + status.name());

          if (context != null) {
            synchronized (context) {

              context.notify();
            }
          }
        }
      }
    private static class MessageSender implements Runnable {
        public void run()  {
          try {


                // Determmina el número de piezas fabricadas y sirve como referencia en los procesos de control del sistema
                int contador_envios = 0;

                // Array que contiene los datos de entrada para el modelo de sensor virtual (la primera columna es el valor real, util para control)
                double sv[][]= {
                        {1470, 25.54, 6.38,     25.55,  476.95, 600},
                        {1480,  27.04,  6.38    ,       25.43,  478.04,  600},
                        {1490,  29.16,  6.41,   25.43,  469.75,  600},
                        {1500,  29.66,  6.41,   25.43,  468.71,  600},
                        {1510,  31.41,  6.41    ,       25.43,  464.91,  600},
                        {1520,  33.32,  6.41    ,       25.43,  455.11,  600},
                        {1520,  33.07,  6.41    ,       25.43,  439.44,  600},
                        {1520,  32.57,  6.41,   25.43,  475.53,  600},
                        {1510,  31.66,  6.34,   25.3    ,       471.51,  600},
                        {1500,  30.16,  6.25    ,       25.3,   458.4,   600},
                        {1490,  28.91,  6.16    ,       25.3,   461.86,  600},
                        {1490,  27.66,  6.22,   25.3    ,       450.56,  484.25},
                        {1480,  26.29,  6.28    ,       25.3,   441.77,  490.11},
                        {1480,  24.91,  6.31    ,       25.3,   437.53,  490.11},
                        {1480,  23.91,  6.34,   25.3    ,       427.35,  493.04},
                        {1480,  23.66,  6.41,   25.3,   416.71,  495.24},
                        {1480,  23.79,  6.38    ,       25.3,   400.94,  496.7},
                        {1480,  23.79,  6.34    ,       25.3,   387.12,  497.44},
                        {1490,  23.91,  6.38,   25.3    ,       369.3,   498.9},
                        {1510,  24.29,  6.38,   25.3    ,       284.76,  495.24}
                };

                // Array que contiene los datos de entrada para el modelo de energía (la última columna es el valor real, util para control)


                double en[][]= {
                        {10900,         1169,   4,      4,      67,     2,      1,      1},
                        {48,    5951,   2,      2,      22,     1,      1,      2},
                        {12,    2096    ,2      ,       3,      49,     1,      2,      1},
```

```
                                {42,    7882,   2,      4,      45,     1,      2,
1},
                                {24,    4870,   3,      4,      53,     2,      2,
2},
                                {36,    9055,   2,      4,      35,     1,      2,
1},
                                {24,    2835,   3,      4,      53,     1,      1,
1},
                                {3,     6948,   2,      2,      35,     1,      1,
1},
                                {12,    3059,   2,      4,      61,     1,      1,
1},
                                {30,    5234,   4,      2,      28,     2,      1,
2},
                                {12,    1295,   3,      1,      25,     1,      1,
2},
                                {48,    4308,   3,      4,      24,     1,      1,
2},
                                {12,    1567,   1,      1,      22,     1,      1,
1},
                                {24,    1199    ,       4,      4,      60,     2,
1,      2},
                                {15,    1403,   2,      4,      28,     1,      1,
1},
                                {24,    1282,   4,      2,      32,     1,      1,
2},
                                {24,    2424    ,       4,      4,      53,     2,
1,      1},
                                {30,    8072,   2,      3,      25,     3,      1,
1},
                                {24,    12579,  4,      2,      44,     1,      1,
2},
                                {9      ,       2134,   4,      4,      48,     3,
1,      1}
                        };
```

// Array que contiene los datos de entrada para el modelo de calidad (la primera columna es el valor real, util para control)

```
                        double ca[][]= {
                                {0      ,       265.1,  10,             197.4,  244.7,
110},
                                {0      ,       161.6,  13.7,   195.5,  254.4,  123},
                                {0      ,       243.4,  12.2,   121.2,  162.6,  114},
                                {0,     299.4,  6.6,            61.9    ,       196.9,
7},
                                {0,     166.7,  10.1,   148.3,  186.9,  113},
                                {0,     223.4,  6.3,            220.6,  203.9,  9},
                                {1      ,       129.1,  12.7,   228.5,  208.8,  137},
                                {1,     332.9,  5.4,            317.8,  160.6,  67},
                                {0      ,       196.4,  13.8,   280.9,  89.3,   139},
                                {0      ,       190.7,  8.1,            218.2,  129.6,
114},
                                {0      ,       189.7,  10,             212.8,  165.7,
66},
                                {1,     62.4,   5.7,            169.9,  209.6,  89},
                                {0      ,       183,            9.5,            72.9,
181.8,  112},
                                {0,     110.4,  7.7,            137.3,  189.6,  103},
                                {0,     81.1,   10.3,   245.2,  237,            86},
                                {0,     124.3,  15.5,   277.1,  250.7,  76},
                                {0,     213,            9.5,            191.1,  182.7,
115},
                                {0      ,       134.3,  14.7,   155.5,  102.1,  73},
                                {0,     190,            6.3,            258.2,  181.5,
109},
                                {0      ,       119.3,  11.1,   215.1,  178.7,  117}
```

```
                };
                 // Creamos unos contadores aleatorios para que recorran el array y muestren
valores aleatorios


            while (true) {
                        System.out.println("Número de mensaje enviado "+ contador_envios);
                        contador_envios ++ ;

                        // Creamos variables enteras de 0 a 20 que nos permitiran recorrer las
filas de las matrices aleatoriamente
                          int countsv = (int) (Math.random()*20);
                          int counten = (int) (Math.random()*20);
                          int countca = (int) (Math.random()*20);

                        TelemetryDataPoint telemetryDataPoint = new TelemetryDataPoint();

                        // Envío datos sv

                        telemetryDataPoint.deviceId="Syts-01";
                        telemetryDataPoint.CE = sv[countsv][0];
                        telemetryDataPoint.Temp_CE = sv[countsv][1];
                        telemetryDataPoint.PH = sv[countsv][2];
                        telemetryDataPoint.Temp_PH = sv[countsv][3];
                        telemetryDataPoint.Turbidez = sv[countsv][4];
                        telemetryDataPoint.Voltaje = sv[countsv][5];
                        telemetryDataPoint.DisPlanta="CANTIL";


                        // Envío datos calidad

                        telemetryDataPoint.CA_Cambio= ca[countca][0];
                        telemetryDataPoint.CA_MOSE = ca[countca][1];
                        telemetryDataPoint.CA_MOS5= ca[countca][2];
                        telemetryDataPoint.CA_MES1= ca[countca][3];
                        telemetryDataPoint.CA_MES2= ca[countca][4];
                        telemetryDataPoint.CA_MES3= ca[countca][5];
                        telemetryDataPoint.contador_piezas_total = contador_envios;


                        // Envío datos energía

                        telemetryDataPoint.Col2 = en[counten][0];
                        telemetryDataPoint.Col5 = en[counten][1];
                        telemetryDataPoint.Col8 = en[counten][2];
                        telemetryDataPoint.Col11 = en[counten][3];
                        telemetryDataPoint.Col13 = en[counten][4];
                        telemetryDataPoint.Col16 = en[counten][5];
                        telemetryDataPoint.Col18 = en[counten][6];
                        telemetryDataPoint.Col21 = en[counten][7];

                        // Serializamos el objeto para crear un mensaje que contenga la
información
                String msgStr = telemetryDataPoint.serialize();
                Message msg = new Message(msgStr);
                msg.setMessageId(java.util.UUID.randomUUID().toString());
                System.out.println("Sending: " + msgStr);

                  // Creamos una respuesta del IotHub que nos permita identificar el estado de la
recepción de los mensajes
                Object lockobj = new Object();
                EventCallback callback = new EventCallback();
                client.sendEventAsync(msg, callback, lockobj);

                synchronized (lockobj) {
                  lockobj.wait();
                }
```

```
                    // Tiempo determinado entre envíos en mlsg
                    Thread.sleep(2000);
                  }
                } catch (InterruptedException e) {
                  System.out.println("Finished.");
                }
              }
            }

        public static void main( String[] args ) throws IOException, URISyntaxException {
            //connString = (args [0]).toString();
            client = new DeviceClient(connString, protocol);
            client.open();
            //System.out.println(connString);
            MessageSender sender = new MessageSender();

            ExecutorService executor = Executors.newFixedThreadPool(1);
            executor.execute(sender);

            System.out.println("Press ENTER to exit.");
            System.in.read();
            executor.shutdownNow();
            client.closeNow();
          }

}
```